Recent Developments in Floorplan Representations

Katsuhisa YAMANAKA*

Department of Electrical Engineering and Computer Science, Iwate University, Morioka 020-8551, Japan

A floorplan is a partition (dissection) of a rectangle into smaller rectangles by horizontal and vertical line segments such that no four rectangles meet at the same point. Floorplans are used to design the layout of very-large-scale integration (VLSI) circuits. Since modern VLSI circuits are extremely large, it is necessary to design compact floorplans (VLSI layouts). In 2004, Feng et al. [8] surveyed ways of representing floorplans. However, over the past decade, various new methods have been developed, and in this paper, we survey these recent developments in floorplan representations.

KEYWORDS: floorplans, data structures, compact representations, information-theoretic lower bound, mosaic floorplans

1. Introduction

Designing an efficient algorithm for a given problem is the most fundamental and important challenge in computer science. In order to do so, it is also necessary to design appropriate data structures for the various objects to which the algorithms are applied. Once we have designed a data structure for the target objects, we then must determine how it can be evaluated. Typically, data structures are evaluated by the maximum number of bits required to represent an object. This leads to the question: how compact can we make the representation of a given object? In particular, consider a class (set) \( C \) of objects. We wish to represent an object \( e \in C \) by a binary code \( S_e \) so that \( S_e \) can be decoded to reconstruct \( e \). In other words, we wish to be able to use \( S_e \) to distinguish \( e \) from any other objects in \( C \). Therefore, in any coding scheme, we must assign a distinct binary code to each object in \( C \), and hence the average length of \( S_e \) is at least \( \log_2 |C| \) bits, which is called the information-theoretic lower bound [13, 16].

It is easy to design a binary code for \( C \) that attains the information-theoretic lower bound, as follows. By using any algorithm that generates all objects in \( C \), we code the \( k \)-th generated object into the binary representation of \( k \). Clearly, this code attains the information-theoretic lower bound of \( C \). However, the time required to code and decode may be extremely large, because the generating algorithm must be repeated each time. Unfortunately, many applications require an efficient running time. Hence, it is very desirable to design a code that can efficiently code and decode.

In the next subsection, we will give an example of this.

1.1 Example of Coding and Its Lower Bound

We begin by giving an example of the representation of ordered trees.

For ordered trees with \( n \) vertices, we have the following code \( S_T \) of length \( 2(n - 1) \) bits [19, 20]. Given an ordered tree \( T \), we begin at the root and perform a depth-first traversal. Edges encountered as we go down are encoded with 0, and those encountered as we go up are encoded with 1. See Fig. 1 for examples. Thus, any ordered tree \( T \) with \( n \) vertices can be represented by a code \( S_T \) of length \( 2(n - 1) \) bits. Clearly, the coding and decoding can each be done in linear time.

![Fig. 1. Examples of ordered trees with four vertices, and their codes.](image-url)
Can we construct a better code? The number of ordered trees with \( n \) vertices is known as the Catalan number \( C_n \), where \( C_n = \frac{1}{n+1} \binom{2n}{n} \) [28]. For example, the number of ordered trees with four vertices is \( C_4 = 5 \), as depicted in Fig. 1. Since each ordered tree with \( n \) vertices must correspond to a distinct code, the average length of \( S_T \) is at least \( \log_2 C_n \). Because the Catalan number can be written as [11, p. 495]
\[
C_n = \frac{4^n}{(n+1)\sqrt{\pi n}} \left( 1 - \frac{1}{8n} + \frac{1}{128n^2} + \frac{5}{1024n^3} - \frac{21}{32768n^4} + O(n^{-5}) \right),
\]
we need at least \( \log_2 C_{n-1} = 2n - o(n) \) bits for \( S_T \). Thus, the length of \( S_T \) above is asymptotically optimal.

We remark that there are several known asymptotically optimal codes for ordered trees: LOUDS (level-order unary degree sequence) [16], DFUDS (depth-first unary degree sequence) [4], and tree covering [10].

### 1.2 Coding Floorplans

In this paper, we will focus on the data structures for floorplans. A floorplan is a partition (dissection) of a rectangle into smaller rectangles, called inner faces, by horizontal and vertical line segments such that no four rectangles meet at the same point. For instance, Fig. 2 shows twenty-four floorplans with four inner faces.

<table>
<thead>
<tr>
<th>Type of floorplans</th>
<th>Lower bounds (bits)</th>
<th>Coding idea</th>
<th>Required memory (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floorplan</td>
<td>( 3.53f )</td>
<td>Removing sequence (Section 3)</td>
<td>( 5f - 5 ) [33]</td>
</tr>
<tr>
<td></td>
<td>for floorplans</td>
<td>Staircase (Section 4)</td>
<td>( 4f - 4 ) [31]</td>
</tr>
<tr>
<td></td>
<td>without edge lengths</td>
<td>(with edge lengths)</td>
<td>( 4f - 4 + (f + 1) \log_2 L ) [23]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DFS (Section 5)</td>
<td>( 6f + 5 ) [34]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dual graph (Section 6)</td>
<td>( 4f - B - 3 ) [23]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mosaic floorplan</td>
<td>( 5f + o(f) ) [34]</td>
</tr>
<tr>
<td>Mosaic floorplan</td>
<td>( 3f - o(f) )</td>
<td>Staircase (Section 7)</td>
<td>( 3f - 3 ) [12]</td>
</tr>
<tr>
<td>Rectangle packing</td>
<td>-</td>
<td>Sequence-pair (Section 8)</td>
<td>( 2f \log_2 f ) [21]</td>
</tr>
</tbody>
</table>

Can we construct a better code? The number of ordered trees with \( n \) vertices is known as the Catalan number \( C_{n-1} \), where \( C_n = \frac{1}{n+1} \binom{2n}{n} \) [28]. For example, the number of ordered trees with four vertices is \( C_4 = 5 \), as depicted in Fig. 1. Since each ordered tree with \( n \) vertices must correspond to a distinct code, the average length of \( S_T \) is at least \( \log_2 C_{n-1} \). Because the Catalan number can be written as [11, p. 495]
\[
C_n = \frac{4^n}{(n+1)\sqrt{\pi n}} \left( 1 - \frac{1}{8n} + \frac{1}{128n^2} + \frac{5}{1024n^3} - \frac{21}{32768n^4} + O(n^{-5}) \right),
\]
we need at least \( \log_2 C_{n-1} = 2n - o(n) \) bits for \( S_T \). Thus, the length of \( S_T \) above is asymptotically optimal.

We remark that there are several known asymptotically optimal codes for ordered trees: LOUDS (level-order unary degree sequence) [16], DFUDS (depth-first unary degree sequence) [4], and tree covering [10].

### 1.2 Coding Floorplans

In this paper, we will focus on the data structures for floorplans. A floorplan is a partition (dissection) of a rectangle into smaller rectangles, called inner faces, by horizontal and vertical line segments such that no four rectangles meet at the same point. For instance, Fig. 2 shows twenty-four floorplans with four inner faces.

An application of this is very-large-scale integration (VLSI) floorplanning [14, 24], where floorplans are used to model the layout of VLSI circuits. Each inner face corresponds to a module that is a functional entity of a chip. If two modules have interconnections, then their corresponding inner faces must be adjacent, that is, the two inner faces must have a common boundary. Since modern VLSI circuits are extremely large, it is desirable to be able to represent them in a compact way.

There has been much research on the best data structures to use with floorplans. In 2004, Feng et al. [8] surveyed floorplan representations. However, over the past decade, various new representations have been developed, and in this paper, we will consider recent developments in floorplan representations. Table 1 summarizes the representations discussed in this paper. For each representation, the table shows: (1) the type of floorplans, (2) the lower bound on required memory space, (3) the coding idea, and (4) the memory required by the code.
1.3 Organization

In this paper, we consider three types of floorplans: (normal) floorplans, mosaic floorplans, and rectangle packings. (Their formal definitions will be given below.) In Sections 3–6, we survey representations for floorplans. More precisely, Section 3 provides a code based on the “removing sequence.” Section 4 provides a code based on the “staircase” of a floorplan; by modifying this code, we obtain a code for a floorplan with edge lengths. Section 5 provides two codes based on the depth-first traversal of a floorplan. Section 6 provides a code based on the dual graph of a floorplan, and which supports adjacency and queries of the degree of inner faces. In Section 7, we provide a code for a mosaic floorplan; this code is extremely compact, and its length is asymptotically equal to the information-theoretic lower bound. Section 8 provides a representation for a rectangle packing, which is a set of rectangles placed on a plane. Finally, Section 9 presents our conclusions.

2. Preliminaries

In this section, we give the basic definitions and observations for (normal) floorplans, which will be discussed in Sections 3–6. We also give two naive representations of floorplans in Section 2.2. We then give the information-theoretic lower bound for representing a (normal) floorplan in Section 2.3.

2.1 Floorplans

In the Introduction, we defined the floorplan of a rectangle. A floorplan can be seen as a plane drawing of a graph of maximum degree three, in which every face is a rectangle [3,33,34], where a plane drawing of a graph is an embedding of the graph in the plane so that no edges intersect except at a vertex to which they are both incident. Therefore, graph-theoretic terminology is sometimes used, as follows.

Let $R$ be a floorplan. Each bounded rectangle is called an inner face. The unbounded region is called the outer face. A vertex of $R$ is a point that is a corner of some rectangle (inner face). There are four vertices for each rectangle. An edge is a line segment connecting two vertices such that it contains no other vertex. We denote by $(u,v)$ the edge connecting the two vertices $u$ and $v$. The contour of a face is the clockwise cycle formed by the vertices and edges on the boundary of the face. For a face $F$, we call its upper horizontal maximal line segment the north line segment of $F$. In a similar way, we define the east, south, and west line segments. Note that the north, east, south, and west line segments of the outer face are, respectively, the uppermost horizontal, rightmost vertical, lowest horizontal, and leftmost vertical line segments of a floorplan.

We next introduce the concept of an “isomorphism” of a floorplan. The base of a floorplan is the south line segment of the outer face, and we will assume that the base of a floorplan is drawn as the lowest horizontal line segment. Two faces $F_1$ and $F_2$ are ns-adjacent (north–south adjacent) if they share a horizontal line segment on their contours. In Fig. 3(a), $F_1$ and $F_2$ are ns-adjacent. Two faces $F_1$ and $F_2$ are ew-adjacent (east–west adjacent) if they share a vertical line segment on their contours. In Fig. 3(b), $F_1$ and $F_2$ are ew-adjacent. If there is a one-to-one correspondence between two floorplans $R_1$ and $R_2$ such that all ns- and ew-adjacencies are preserved, and the base of $R_1$ corresponds to that of $R_2$, then we say that $R_1$ and $R_2$ are isomorphic. The two floorplans in Figs. 4(a) and (b) are isomorphic, although the lengths of their edges are different. However, the two floorplans in Figs. 4(a) and (c) are not isomorphic, because, for example, the face $d$ is ew-adjacent to the face $i$ in Fig. 4(c) but not in Fig. 4(a). If we rotate the floorplan in Fig. 3(a), then we obtain the same floorplan as in Fig. 3(b). However, the two floorplans are non-isomorphic, since the bases do not correspond. Based on this isomorphism, we can observe that there are twenty-four floorplans that have four inner faces, as shown in Fig. 2.

If two faces $F_1$ and $F_2$ are ns- or ew-adjacent, we say that $F_1$ is a neighbour of $F_2$ and vice versa. $F_1$ is a west neighbour of $F_2$ if $F_1$ is located to the west of $F_2$. We denote by west$(F)$ the set of west neighbours of a face $F$. We define the east, north, and south neighbours of $F$ in a similar way and denote the number of them by east$(F)$, north$(F)$, and south$(F)$, respectively. In Fig. 4(c), the west neighbour of $e$ is $d$, thus west$(e) = \{d\}$. The east neighbour of $e$ is $g$, thus east$(e) = \{g\}$. The north neighbours of $e$ are $a$ and $b$, thus north$(e) = \{a,b\}$. The south neighbours of $e$ are $i$, $j$, $k$, and $l$, thus south$(e) = \{i,j,k,l\}$. To emphasize the floorplan $R$, the set east$(F)$ is sometimes denoted by east$(R,F)$; the same applies to west$(R,F)$, south$(R,F)$, and north$(R,F)$.

We can assume without loss of generality there are only four vertices of degree two, and they appear on the corners.
of the outer face; otherwise, such a degree-two vertex is redundant. Thus, every vertex except those vertices on the corners of the outer face has degree three, as is standard in the literature [14, 22, 35]. We adopt this convention to simplify the discussion.

Let \( n \) be the number of vertices in a floorplan, let \( m \) be the number of edges, and let \( f \) be the number of inner faces. Since every vertex of a floorplan is of degree two or three, a floorplan contains \( n - 4 \) vertices of degree three and four vertices of degree two (at the four corners of the outer face). Hence we have \( 2m = 3(n - 4) + 8 \). This equation and Euler’s formula \( n - m + (f + 1) = 2 \) implies that

\[
\begin{align*}
n = 2(f + 1) \\
m = 3f + 1.
\end{align*}
\]

A vertex of degree three is \( w \)-missing (west missing) if it has line segments only to the top, bottom, and right. We denote the number of \( w \)-missing vertices as \( n_w \). In a similar way, we define \( e \)-missing (east missing), \( n \)-missing (north missing), and \( s \)-missing (south missing), respectively. Each \( w \)-missing vertex is the left end of some maximal horizontal line segment, and each \( e \)-missing vertex is the right end of some maximal horizontal line segment. Hence \( n_w = n_e \) holds. Similarly, we have \( n_N = n_S \). Therefore, we have the following equation:

\[
n_E + n_N = \frac{n - 4}{2}.
\]

### 2.2 Naive Representation of Floorplans

We now introduce two naive representations of floorplans.  

An adjacency matrix and an adjacency list are standard representations of graphs. We can use these representations to store the graph structures of floorplans. Recall that an adjacency matrix representation contains information about the adjacencies of any two vertices. If two vertices \( i, j \) are adjacent in a graph, then the value of \( i \)-th row and \( j \)-th column is one; otherwise, it is zero. On the other hand, an adjacency list stores as a list all the neighbours of each vertex.

If we represent a floorplan with \( n \) vertices, \( m \) edges, and \( f \) inner faces with an adjacency matrix representation, then we need an \( n \times n \)-matrix in which each entry is either zero or one. Hence, the required memory is \( \Theta(n^2) \) bits. Next, we consider representing the floorplan by an adjacency list. Note that we need \( \Theta(\log_2 n) \) bits to represent the index of each vertex, and hence we use \( \Theta(m \log_2 n) \) bits in total.

Thus, a floorplan can be represented easily by a binary code with either \( \Theta(n^2) \) or \( \Theta(m \log_2 n) \) bits. This leads naturally to two questions: (1) What is the minimum number of bits required to represent a floorplan? (2) Are naive representations compact? In the next subsection, we will introduce lower and upper bounds on the number of bits required to represent a floorplan.

### 2.3 The Number of Floorplans and the Information-Theoretic Lower Bound

For a specified number of inner faces, how many floorplans are there? If we know the number of floorplans with \( f \) inner faces, we can obtain the information-theoretic lower bound of floorplans with \( f \) inner faces. There has been much research on counting floorplans and on bounding the number of floorplans.

Let \( F(f) \) be the number of floorplans that are distinct (up to isomorphism) and have \( f \) inner faces. There have been several studies to count \( F(f) \). Nakano [22] provided an efficient algorithm with which to enumerate all the floorplans with \( f \) inner faces. The algorithm generates each floorplan in \( \Theta(1) \) time, on average. By implementing this algorithm, Yoshii and Nakano [36] reported the exact number of floorplans with \( f \) inner faces for \( f \leq 13 \). Amano et al. [3] proposed a faster algorithm for counting \( F(f) \) and reported the exact number of floorplans with \( f \) inner faces for \( f \leq 30 \). The

*Precisely, a floorplan is a plane drawing of a graph. Thus, it is necessary to store information other than the graph structure in order to represent a floorplan. However, in order to simplify this section, we will ignore such additional information.*
running times of both of these algorithms are exponential in the number of inner faces. Surprisingly, Inoue et al. [15] proposed a polynomial-time algorithm to count $F(f)$. Using this algorithm, they reported the exact number of floorplans with $f$ inner faces for $f \leq 3000$. Their results are interesting theoretically, and the algorithm is extremely fast.

Amano et al. [3] showed the upper and lower bounds of $F(f)$. They showed that $F(f) = \Omega(2^{3.53f})$ and $F(f) = O(2^{3.75f})$. This lower bound implies that any representation of a floorplan with $f$ inner faces requires at least $3.53f$ bits, on average. Inoue et al. [15] improved the upper bound to $F(f) = O(2^{3.64f})$, and then Fujimaki et al. [9] proposed a tighter upper bound of $F(f) = O(2^{3.75f})$.

Theorem 2.1 ([3.9]). $F(f) = \Omega(2^{3.53f})$ and $F(f) = O(2^{3.75f})$ hold.

From the above theorem, we obtain the lower and upper bounds of the information-theoretic lower bound of floorplans with $f$ inner faces. It can be observed that we need at least $3.53f$ bits on average to represent such floorplans. In the following sections, we will consider ways to represent floorplans with fewer bits by using binary codes. Throughout this paper, we assume that the end-of-file is prepared as a special character, and none of the codes in this paper contain an end-of-file.

3. Coding Based on the Removing Sequence

In this section, we provide a code for floorplans that is based on the “removing sequence.” First, we define the removing sequence of a floorplan, and then we explain the related code. With this method, any floorplan with $f$ inner faces can be represented by $5f - 5$ bits.

3.1 Removing Sequence

We now define the removing sequence of a floorplan. Roughly speaking, the removing sequence of a floorplan is a sequence of floorplans that can be obtained by sequentially shrinking the leftmost and uppermost inner faces. (See Fig. 5 for an example.) We now give a formal definition.

We assume that $R$ is a floorplan with $f > 1$ inner faces. Let $F$ be the inner face of $R$ having the upper-left vertex of the outer face of $R$, and hence $\text{north}(F) = \emptyset$ and $\text{west}(F) = \emptyset$. We call such a leftmost and uppermost inner face the first face of $R$. The first faces of the floorplans are shaded in Figs. 5–7. Let $v$ be the lower-right vertex of $F$, as illustrated in Fig. 6. The first face $F$ is upward removable if $v$ is e-missing; see Fig. 6(a). Otherwise $v$ must be s-missing, and the first face $F$ is said to be leftward removable. (See Fig. 6(b).) Since $R$ has two or more inner faces, the first face of $R$ is either upward removable or leftward removable. If the first face $F$ is upward removable, then we can obtain another floorplan with one less inner face by continually shrinking $F$ into the north line segment of the outer face of $R$, while preserving the width of $F$ and enlarging the south neighbours of $F$, as shown in Fig. 7. Similarly, if $F$ is leftward removable, then we can obtain another floorplan with one less inner face by continually shrinking $F$ into the west line segment of $R$ while preserving the height of $F$. After we shrink the first face from $R$, the resulting floorplan has one less inner face. We denote such a floorplan by $P(R)$. Thus we can define the floorplan $P(R)$ for each floorplan $R$ with two or more inner faces.
Given a floorplan $R$ with $f$ inner faces, by repeatedly shrinking the first faces, we can obtain a unique sequence $R, P(R), P(P(R)), \ldots$ of floorplans that eventually ends with a floorplan that has only one inner face. See Fig. 5 for an example; note that the first faces are shaded. We call such a sequence the removing sequence of $R$. Note that the removing sequence of a floorplan with $f$ inner faces consists of $f$ floorplans.

Using the removing sequence, we can obtain a naive code, as follows. Let $RS = (R_1, R_{i-1}, \ldots, R_i)$ be the removing sequence of $R$, and hence $R_1 = R$ and $R_{i-1} = P(R_i)$. We denote by $F_i$ the first face of $R_i$ for each $i = f, f - 1, \ldots, 3, 2$. Each first face $F_i$ has $\{\text{south}(F_i)\}$ south neighbours and $\{\text{east}(F_i)\}$ east neighbours. Recall that $R_{i-1} = P(R_i)$ is the floorplan obtained by removing the first face $F_i$ of $R_i$. Therefore, given $R_{i-1} = P(R_i)$, if we know (1) whether the first face $F_i$ of $R_i$ is upward removable or leftward removable, and (2) the two values $\{\text{south}(F_i)\}$ and $\{\text{east}(F_i)\}$, then we can reconstruct $R_i$ from this information. The information about (1) for $F_i$ is represented by 1 bit; 0 indicates that $F_i$ is upward removable, and 1 indicates that $F_i$ is leftward removable. The information about (2) for $F_i$ is represented by $2 \log_2 f$ bits, because $\{\text{south}(F_i)\} \leq f$ and $\{\text{east}(F_i)\} \leq f$, and hence each of these values can be represented by $\log_2 f$ bits.

Therefore, we can represent each $F_i$, $f \geq i \geq 2$, with $1 + 2 \log_2 f$ bits in total. Since $(f - 1)$ first faces $F_f, F_{f-1}, \ldots, F_2$ need to be stored, this naive code requires $(f - 1)(1 + 2 \log_2 f)$ bits in total. In the next subsection, we will provide a more compact code for floorplans.

### 3.2 Code for Floorplans

In Section 3.1, we explained a naive code based on the removing sequence of a floorplan $R$ with $f$ inner faces. In the naive code, we used $2 \log_2 f$ bits to represent the information (2) for each floorplan, that is, to represent the numbers of south neighbours and east neighbours of the first face for each floorplan in the removing sequence of $R$. Therefore, we used $2(f - 1) \log_2 f$ bits in total to represent the information (2). In this subsection, we show that the information (2) can be represented by $4(f - 1)$ bits in total.

We begin by providing some definitions. An inner face $F$ of a floorplan is U-active if north($F$) $\neq \emptyset$, that is, no inner face is located above $F$. Therefore, a U-active inner face touches the north line segment. For convenience, we will also regard the outer face as U-active. Let $RS = (R_1, R_{i-1}, \ldots, R_i)$ be the removing sequence of $R$, and denote by $F_i$ the first face of $R_i$ for each $i = f, f - 1, \ldots, 2$. We partition $RS$ into two subsequences $RS_U$ and $RS_L$, as follows. The subsequence $RS_U = (R_1^U, R_2^U, \ldots, R_f^U)$ consists of the floorplans in $RS$, preserving the order, whose first faces are upward removable; while the subsequence $RS_L = (R_1^L, R_2^L, \ldots, R_f^L)$ consists of the floorplans in $RS$, preserving the order, whose first faces are leftward removable. Note that $f_U + f_L = f - 1$ holds, since the floorplan with one inner face is contained in neither $RS_U$ nor $RS_L$.

Now we construct our code, as in the following five parts. Part 1 encodes the information (1), that is, whether the first face of each $R_i$, $f \geq i \geq 2$, is upward removable or leftward removable. Parts 2 and 3 encode the information (2) for $RS_U$, that is, the values $\{\text{south}(F_i)\}$ for each $R_i$ in $RS_U$, respectively. Parts 4 and 5 encode the information (2) for $RS_L$, that is, the values $\{\text{south}(F_i)\}$ and $\{\text{east}(F_i)\}$ of $F_i$ for each $R_i$ in $RS_L$, respectively.

**Part 1:** This part encodes whether the first face of each $R_i$, $f \geq i \geq 2$, is upward movable or leftward movable. For $i = 2, 3, \ldots, f$, the $i$-th bit is 0 if the first face of $R_i$ is upward removable, and it is 1 otherwise. Therefore, Part 1 requires $f - 1$ bits in total.

**Part 2:** Part 2 encodes each value $\{\text{south}(F_i)\}$ of the first face $F_i$ of $R_i \in RS_U$. We can observe that the value $\{\text{south}(F_i)\}$ is equal to the number of inner faces that are U-active in $R_{i-1} = P(R_i)$ but that are not U-active in $R_i$. Furthermore, once an inner face becomes U-active, it remains U-active until it is removed. Therefore, each inner face becomes U-active exactly once, and hence

$$\sum_{R_i \in RS_U} |\text{south}(F_i)| = f - 1.$$  

Note that, when we remove a leftward movable face, no face newly becomes U-active.

We encode each $\{\text{south}(F_i)\}$ for $R_i \in RS_U$ as a unary code, where a unary code of a positive integer $k$ is $k - 1$ consecutive 0s followed by one 1. For example, we code $\{\text{south}(F_i)\} = 3$ as “00011.” Since the first face $F_i$, $2 \leq i \leq f$, always has at least one south neighbour, $|\text{south}(F_i)| \geq 1$ holds for each $i$. After we concatenate those codes, we finally append zeroes so that the length of Part 2 becomes exactly $f - 1$ bits. We can easily encode each $\{\text{south}(F_i)\}$ for all $R_i \in RS_U$ from the code.

**Part 3:** Part 3 encodes each value $\{\text{east}(F_i)\}$ of the first face $F_i$ of $R_i \in RS_U$. Note that the sum of the $\{\text{east}(F_i)\}$ may be large, since two or more floorplans in $RS_U$ contain the same inner face that is an east neighbour of a U-active inner face. In other words, $\text{east}(F_i) \cap \text{east}(F_j) \neq \emptyset$ may hold for two floorplans $R_i, R_j \in RS_U$, and hence, we do not directly encode each $\{\text{east}(F_i)\}$ in a unary code.

We provide the code for Part 3, together with our idea; see also Fig. 8. Let $F_{SE}$ be the easternmost south neighbour of the first face $F_i$ of $R_i \in RS_U$, and hence the eastern boundaries of $F_i$ and $F_{SE}$ are on the same vertical line segment of $R_i$. Observe that exactly one inner face $F_{SE}$ is contained in $\text{east}(R_i, F_i) \cap \text{east}(R_j, F_{SE})$. By removing $F_i$ from $R_i$, we obtain the floorplan $R_{i-1} = P(R_i)$. Then, the set $\text{east}(R_i, F_i)$ can be expressed as follows:
Recent Developments in Floorplan Representations

4. Staircase-Based Code

In this section, we explain a compact code for a floorplan, which was given by Takahashi et al. [31]. This code is based on the “staircase” of a floorplan, which was originally proposed in [15]. The notion of the staircase had a strong influence on the research for floorplans. Recently, several algorithms based on the staircase have been proposed for determining the bounds of the number of floorplans [9], counting the number of floorplans [15], coding floorplans [31], and coding mosaic floorplans [12, 30].
4.1 Staircase

Let $F$ be an inner face of a floorplan $R$. The upper-right area of $F$ is the area bounded by:
(a) the vertical half-line that is drawn upward from the upper-left vertex of $F$,
(b) the north line segment of $F$,
(c) the east line segment of $F$, and
(d) the horizontal half-line that is drawn rightward from the lower-right vertex of $F$.

Figure 9 illustrates the upper-right area of $F$, which is shaded. We defined the upper-right area of $F$ to be open, and hence it does not contain boundaries. An inner face $F$ is clear if the upper-right area of $F$ intersects no other inner face of $R$. If the vertical half-line corresponding to (a) overlaps the east line segment of an inner face $F_U$, then we say that $F_U$ is an upward predecessor of $F$; see Fig. 9 for an example. Similarly, if the horizontal half-line corresponding to (d) overlaps the north line segment of an inner face $F_R$, then we say that $F_R$ is a rightward predecessor of $F$; see Fig. 9 for an example. A clear inner face is ready if it has neither an upward predecessor nor a rightward predecessor. The top-ready inner face of $R_i$ is the ready inner face whose lower-left vertex is highest among all of the ready inner faces in $R_i$. Then, we define $R_{i-1}$ as the floorplan obtained by deleting the top-ready inner face of $R_i$.

For a floorplan $R$ with $f$ inner faces, we say that $SS = (R_f, R_{f-1}, \ldots, R_1)$ is the staircase sequence of $R$ if the following conditions hold (see Fig. 10):

1. $R_f = R$; and
2. for each $i = f, f-1, \ldots, 2$, we define $R_{i-1}$ in $SS$ as a sub-floorplan obtained from $R_i$ by deleting an inner face of $R_i$.

In Section 3, we used the term “remove.” To distinguish between the two operations, we use the term “delete” in this section.
$R_i$. (We will show below that $R_i$ has at least one ready inner face.)

We define a “staircase” for each sub-floorplan in the staircase sequence $SS = (R_f, R_{f-1}, \ldots, R_1)$ of a floorplan $R$. A staircase of $R_i$ for each $i = f, f-1, \ldots, 1$ is a path along the contour of the outer face from the upper-left vertex to the lower-right vertex. As we can see in Fig. 10, a staircase is monotonically decreasing with respect to the $x$-coordinates (we will prove this below). We will first show that $R_i$ is always constructed from $R_{i+1}$, and then we will show that each $R_i$ in $SS$ has at least one ready inner face, and hence $R_{i-1}$ can always be defined from $R_i$. We first give some properties of a staircase.

**Lemma 4.1** ([23]). Let $R$ be a floorplan with $f$ inner faces, and let $SS = (R_f, R_{f-1}, \ldots, R_1)$ be the staircase sequence of $R$. Then each $R_i$ in $SS$ has at least one ready inner face.

**Proof.** Let $F$ be the inner face whose lower-left vertex is highest among all the inner faces of $R_i$; if $R_i$ has two or more such inner faces, then choose the rightmost one. If $F$ is ready, then we are done. Otherwise, (1) $F$ has some rightward predecessor, or (2) the upper-right area of $F$ intersects with the proper inside of some inner face. (Note that $F$ has no upward predecessor, since $F$ has the highest lower-left vertex.) Let $F'$ be such an inner face with the highest lower-left vertex; if $R_i$ has two or more such inner faces, then choose the rightmost one. By the choice of $F$, the lower-left vertex of $F'$ is lower than the lower-left vertex of $F$, and the lower-right vertex of $F'$ is located to the right of the lower-right vertex of $F$. (Intuitively, $F'$ is located to the lower right of $F$. If $F'$ is ready, then we are done. Otherwise, (1) $F'$ has some rightward predecessor, or (2) the proper inside of some inner face intersects with the upper-right area of $F'$. Let $F''$ be such an inner face for the highest lower-left corner. If $R_i$ has two or more such inner faces, then choose the rightmost one. Again, $F''$ is located to the lower-right of $F'$. By repeating this process a sufficient number of times, we will always find a ready inner face. \square

The above lemma guarantees that $R_{i-1}$ in a staircase sequence $SS$ is always defined from $R_i$ by deleting the top-ready inner face of $R_i$. Since the number of inner faces decreases one by one, we finally obtain the sub-floorplan $R_1$ with exactly one inner face.

Next, we show that a staircase of $R_i$ is a monotonically decreasing path from its upper-left vertex to its lower-right vertex. That is, a staircase consists of an alternating sequence of vertical and horizontal line segments, and it is monotonically decreasing if we regard the vertical and horizontal directions as the $y$-axis and $x$-axis, respectively.

**Lemma 4.2.** Let $R$ be a floorplan with $f$ inner faces, and let $SS = (R_f, R_{f-1}, \ldots, R_1)$ be a staircase sequence of $R$. The staircase of $R_i$ in $SS$ is a monotonically decreasing path from the upper-left vertex to the lower-right vertex of $R_i$.

**Proof.** Proof by induction. See Fig. 10 for an example. For $R_f$, the upper-left vertex and lower-right vertex are uniquely defined, and its staircase consists of the north line segment and the east line segment of $R_f$. Therefore, the staircase of $R_f$ is a stepwise monotonically decreasing path.

Suppose that the claim holds for $R_i$, $i < f$. By Lemma 4.1, we can find the top-ready inner face of $R_i$, which we will call $F_i$. First, we show that the upper-left vertex and the lower-right vertex of $R_{i+1}$ are uniquely defined. If the upper-left vertex of $F_i$ is not the upper-left vertex of $R_i$, then the upper-left vertex does not change. Otherwise the upper-left vertex of $F_i$ is the upper-left vertex of $R_i$. Since $F_i$ is clear, the lower-left vertex of $F_i$ becomes the upper-left vertex of $R_{i+1}$. Thus the upper-left vertex of $R_{i+1}$ is uniquely defined. Similarly, we can show that the lower-right vertex of $R_{i+1}$ is uniquely defined. Since $F_i$ is clear, the staircase of $R_i$ is still a stepwise monotonically decreasing path. \square

Let $R$ be a floorplan with $f$ inner faces, and let $SS = (R_f, R_{f-1}, \ldots, R_1)$ be a staircase sequence of $R$. Then we denote by $F_i$ the top-ready inner face of $R_i$ for each $i = f, f-1, \ldots, 1$. Note that $R_f$ consists of the inner faces $F_1, F_2, \ldots, F_f$. $R_f$ is identical to $R$, and $R_1$ has exactly one inner face $F_1$. Now we define the base height of each $F_i$. First, we introduce two dummy line segments into $R_i$: a vertical line segment that is drawn upward from the upper-left vertex of $R_i$, and a horizontal line segment that is drawn rightward from the lower-right vertex of $R_i$. Then the upper-left vertex and the lower-right vertex of every inner face $F_i$ of $R_i$ is of degree exactly three; see Fig. 11 for an example. We regard the horizontal line segment from the lower-right vertex as the 0-th step. Similarly, we regard the $k$-th horizontal line segment on the staircase from its lower-right vertex as the $k$-th step of the staircase, as shown in Fig. 11. Suppose that $F_i$ is located on the $k$-th bottom horizontal line segment on the staircase of $R_{i-1}$. Then we say that the base height of $F_i$ is $k$. Intuitively, $F_i$ is on the $k$-th step of the staircase of $R_{i-1}$ from its lower-right vertex. For convenience, we regard the rightward line segment starting from the lower-right vertex of $R_{i-1}$ as the 0-th step of the staircase; see Fig. 11 for an example. Next, we classify the top-ready inner faces $F_i$ into four types, defined as follows:

- **Type 0** the upper-left vertex of $F_i$ is $n$-missing, and the lower-right vertex of $F_i$ is e-missing;
- **Type 1** the upper-left vertex of $F_i$ is w-missing, and the lower-right vertex of $F_i$ is e-missing;
- **Type 2** the upper-left vertex of $F_i$ is $n$-missing, and the lower-right vertex of $F_i$ is s-missing; and
- **Type 3** the upper-left vertex of $F_i$ is w-missing, and the lower-right vertex of $F_i$ is s-missing.

Figure 12 illustrates the four types. Note that each top-ready inner face belongs to exactly one of the four types, and $F_1$ is always type 3.
4.2 Staircase-Based Code for Floorplans

Let $R$ be a given floorplan with $f$ inner faces, and let $SS = (R_f, R_{f-1}, \ldots, R_1)$ be a staircase sequence of $R$. We denote by $R_i$ the top-ready inner face of $R$ for $i = f, f-1, \ldots, 1$. Our idea is to encode (i) the type of $R_i$, and (ii) the base height of $R_i$. Now we show that this information is sufficient for reconstructing the original floorplan $R$. If $i = 1$, then we can easily reconstruct $R_1$, since it is just one inner face. Therefore, suppose that $R_i$, $i \geq 1$, is already reconstructed, and we wish to reconstruct $R_{i+1}$ by suitably introducing $R_{i+1}$ into $R_i$. We will show just one case, since the other cases are almost the same. Suppose that $R_i$ is as shown in Fig. 13(a). Since we know the base height of $R_{i+1}$, say 2, we can find the horizontal line segment of $R_i$ on which we are going to introduce $R_{i+1}$. We also know the type of $R_{i+1}$, say type 3, and $R_{i+1}$ is top-ready in $R_{i+1}$. Hence, $R_{i+1}$ can be appended, as shown in Fig. 13(b). It should be noted that, if we append $R_{i+1}$ so that $R_{i+1}$ has an upward predecessor $F_U$ as shown in Fig. 13(c), then $R_{i+1}$ is not ready in $R_{i+1}$. Thus $R_{i+1}$ has no upward predecessor. Similarly $R_{i+1}$ has no rightward predecessor. Therefore $R_{i+1}$ is uniquely introduced to $R_i$ as in Fig. 13(b). Thus we can reconstruct $R_1, R_2, \ldots, R_f = R$ if we have stored information about (i) and (ii). Below, we will give a code for storing the information about (i) and (ii). Note that the type of each top-ready inner face can be represented by two bits, since the number of types is only four. Hence, the information about (ii) can be stored as stated in the following lemma.

**Lemma 4.3** ([23]). *One can encode the base heights of the inner faces into a binary code of length at most $2(f - 1)$.*

**Proof.** Let $R$ be a floorplan, and let $F_i, F_{i-1}, \ldots, F_1$ be the top-ready inner faces of the sub-floorplans in the staircase sequence $SS = (R_f, R_{f-1}, \ldots, R_1)$ of $R$, respectively. We encode the “differences” between two base heights, $F_i$ and $F_{i+1}$. We can observe that (1) the base height of $F_1$ is always 0, (2) the base height of $F_f$ is either 0 or 1, and (3) the difference $d_i$ of two consecutive base heights, where $d_i$ is calculated by subtracting the base height of $F_i$ from the base height of $F_{i+1}$, is $-1$, 0, or a positive integer. For example, $d_8, d_7, \ldots, d_1$ of $R_{10}$ in Fig. 10 are equal to $-1$, respectively. Then we encode $-1$ as “1,” 0 as “01,” 1 as “001,” 2 as “0001,” and continue in this pattern. That is, the code for the difference $d_i$ is the unary code for $d_i + 2$. We encode the base heights of the inner faces as the sequence of the differences encoded by the above method.
Recent Developments in Floorplan Representations

Section 4.2. Adjacency relationships between the inner faces are stored by the code in Section 4.2, which uses \(4\) bits, and we store the information about the edge lengths by another code, which uses \(4\) bits. We then obtain a code for a grid floorplan by concatenating these two codes.

Thus for each difference \(d_i > 0\), we must have the number \(d_i\) of \((-1)\)'s in \((d_1, d_2, \ldots, d_{i-1})\). For the difference \(d_i = 0\), we use two bits. For the difference \(d_i > 0\), we use \(d_i + 2\) bits to represent \(d_i\), and we know that there are \(d_i\) \((-1)\)'s in \((d_1, d_2, \ldots, d_{i-1})\), each of which uses only one bit. Therefore we use \(2d_i + 2\) bits in total to represent these \(d_i + 1\) differences. Thus, on average, we use two bits for each difference, and hence \(2(f - 1)\) bits in total.

Now we estimate the total length of the above code. Suppose that \((d_1, d_2, \ldots, d_{f-1})\) is the sequence of the differences. We will assume that the base height of \(F_f\) is 0. (The other case, in which the base height of \(F_f\) is 1, follows a similar analysis.) The base height of \(F_1\) is always 0. Hence we have

\[
\sum_{i=1}^{f-1} d_i = 0.
\]

Since we know the difference \(d_i\), we can represent a grid floorplan by storing (1) the staircase-based code for floorplans, as in Section 4.2; (2) the width and height of each inner face in binary representation; or (3) the lengths of the binary representations of the widths, the heights, and the number of faces. A naive code uses \(4f - 4 + f[\log_2 W] + f[\log_2 H] + o(f) + o(W) + o(H)\) bits, where \(W\) and \(H\) are, respectively, the maximum width and height of the inner faces.

This subsection presents a more compact code for a grid floorplan [23]. The code uses at most \(4f - 4 + (f + 1)[\log_2 L] + o(f) + o(L)\) bits for a given grid floorplan, where \(L\) is the maximum length of the edges in the floorplan. Note that \(L \leq \max(W, H)\) holds. The coding and decoding algorithms each take \(\Theta(f)\) time. Here we assume that \([\log_2 L]\) is less than the word size. This code for a grid floorplan is based on the staircase-based code in Section 4.2. Adjacency relationships between the inner faces are stored by the code in Section 4.2, which uses \(4f - 4\) bits, and we store the information about the edge lengths by another code, which uses \((f + 1)[\log_2 L] + o(f) + o(L)\) bits. We then obtain a code for a grid floorplan by concatenating these two codes.

Let \(R\) be a grid floorplan with \(f\) inner faces. For a grid floorplan, we can define its staircase sequence in the same way as in Section 4.1. Let \(SS = (R_f, R_{f-1}, \ldots, R_1)\) be the staircase sequence of \(R\), and denote by \(F_i\) the top-ready inner faces of \(R_i\) for each \(i = f, f - 1, \ldots, 1\). We define the hands of each top-ready inner face \(F_i\) as follows. (A hand is a key to the code, as will be explained in this subsection.) If the upper-left vertex \(v\) of \(F_i\) of \(R_i\) is w-missing and hence \(F_i\) is either Type 1 or 3, then \(F_i\) has an upper hand, which is the common vertical line segment of \(F_i\) and the face located to

![An example of a grid floorplan.](image-url)
We used at most \( \log_2 L \) bits to encode the lengths of the hands, using Lemma 4.6. We stored \( f \) and \( \log_2 L \) in gamma code, which use \( 2[\log_2(f + 1)] + 2[\log_2([\log_2 L] + 1)] - 2 \) bits. We thus have the following theorem.

**Lemma 4.6** ([23]). *The number of hands is at most \( f + 1 \).*

**Proof.** We assume that \( R_i \) has \( k \) steps. If \( F_{i+1} \) is type 1 or 2, then \( R_{i+1} \) also has \( k \) steps. If \( F_{i+1} \) is type 0, then \( R_{i+1} \) has \( k - 1 \) steps. If \( F_{i+1} \) is type 3, then \( R_{i+1} \) has \( k + 1 \) steps. The numbers of steps in \( R_f \) and \( R_1 \) are always two. Thus the number of type 0 inner faces is equal to the number of type 3 inner faces (excluding \( F_1 \)). Each type 0 inner face has no hand, each type 1 or 2 inner face has one hand, and each type 3 inner face has two hands. Thus we need one hand for each inner face (excluding \( F_1 \)) on average, and \( F_1 \) has two hands. Thus the number of hands is always \( f + 1 \). 

We now estimate the length of our code. The information about (i) and (ii) is encoded in \( 4f - 4 \) bits, as shown in Section 4.2. We used at most \((f + 1)[\log_2 L] \) bits to encode the lengths of the hands, using Lemma 4.6. We stored \( f \) and \( \log_2 L \) in gamma code, which use \( 2[\log_2(f + 1)] + 2[\log_2([\log_2 L] + 1)] - 2 \) bits. We thus have the following theorem.
One can encode any grid floorplan by using at most 
\[ 4f/C0 + (f + 1) \log_2 L + o(f) + o(L) \] bits.

We may assume that \([\log_2 L]\) is less than the word size. Then, with a suitable data structure, as shown in Section 4.2,
coding and decoding each take \(O(f)\) time. We have the following theorem.

**Theorem 4.8 ([23]).** The coding and decoding algorithms for a grid floorplan each take \(O(f)\) time.

### 5. DFS-Based Code

In this section we give two codes for floorplans (without edge lengths) based on a depth-first traversal. Yamanaka
and Nakano [34] proposed a code based on the depth-first traversal of a given floorplan, as follows: first a given
floorplan is transformed into a tree, and then we encode a depth-first traversal of the tree. The code takes \(6f + 5\) bits,
where \(f\) is the number of inner faces. Recently, Saikawa and Nakano [26] proposed a new compact code by improving
the code in [34]. Their code takes at most \(4f - 4\) bits. Their result is the same length as the staircase-based code. In this
section, we first explain the code proposed by [34], and then explain that of Saikawa and Nakano [26].

#### 5.1 Basic DFS-Based Code

Given a floorplan \(R\) with \(m\) edges and \(f\) inner faces, we first replace the lower-right vertex of each inner face with
two vertices, as depicted in Figs. 16(a) and (b). We then add a new “leg” to each of the two lower vertices of the outer
face, as depicted in Figs. 16(b) and (c). The other vertices remain unchanged. Figure 17 illustrates all the rules for the
replacements. (We note that, in the figures in this section, the vertices in a floorplan are drawn as black or white
circles.) Since we only break each cycle corresponding to an inner face at its lower-right vertex, the resulting graph has
only an outer face and is still connected, so the resulting graph \(R'\) is a tree. Here, we explain a depth-first traversal of \(R'\).
Starting at the upper-left vertex of the outer face, we will traverse the tree \(R'\) in a depth-first manner (with left priority).
It should be noted that we will maintain the embedding of \(R'\), and hence we can appropriately distinguish between
directions during the traversal. Recall that each vertex of a floorplan is of degree at most three. Therefore, when we visit
a vertex, we have only two possibilities for the direction of the next vertex, as shown in Fig. 19. For example, consider
the case in Fig. 19(b). If we trace an edge from left to right, then the next trace will be either down or to the right. If we
assume that the next trace is up, this contradicts the fact that each face in \(R\) has been broken by replacing its lower-right
vertex. Similarly, for the other cases in Fig. 19, we can conclude that the next trace in the depth-first traversal of \(R'\) has
only two possibilities.
We encode the depth-first traversal by representing the next direction from the current vertex. Again, in the case in Fig. 19(b), we store $0$ if the next direction is down and $1$ if the next direction is right. Note that it requires one bit to store the next direction. The depth-first traversal and its code for the floorplan in Fig. 18(a) is shown in Fig. 18(b). Since the first trace is always down, it is not necessary to store this trace. We call this the DFS-based code. The DFS-based code for the floorplan in Fig. 18(a) is

```
00001011011001001010110010101101101110001000100111001001100100110
```

Now we estimate the length of the DFS-based code, as follows. Each edge is traced exactly twice, we use two bits for each edge (except for the first edge, which uses only one bit because we always trace the first edge to the bottom), and we have two dummy edges at the two lower vertices. Therefore, by Eq. (2.2), the DFS-based code takes $2m + 3 = 6f + 5$ bits. Given the code, we can easily reconstruct the original floorplan $R$. We thus have the following theorem.

**Theorem 5.1** ([34]). One can encode any floorplan with $f$ inner faces by using $6f + 5$ bits. The coding and decoding each take $O(f)$ time.

### 5.2 Improvement of the DFS-Based Code

Saikawa and Nakano [26] proposed a more compact code that is based on the DFS-based code in the previous subsection; it is as follows. Let $R$ be a floorplan with $n$ vertices and $f$ inner faces. A bottom edge of $R$ is an edge on the south line segment of $R$. Let $B$ be the number of bottom edges of $R$. For instance, the floorplan in Fig. 18(a) has three bottom edges. We construct a tree $R''$ from $R$, as follows. First we remove the bottom edges in $R$, then break each inner face by replacing its lower-right vertex, as shown in Fig. 17. We note that there is no vertex, as shown in Figs. 17(g) and (h). Let $R''$ be the resulting graph. Since $R''$ is connected and has exactly one face, $R''$ is a tree; see Fig. 18(c) for an example.

Similar to what we did for the DFS-based code, we can encode the depth-first traversal of $R''$ by representing the next direction from the current vertex. Suppose that, in the traversal of $R''$, we trace an edge $e$ and then arrive at $v$. We have the following four possibilities for the next direction.

**Case 1:** $e$ is traced from top to bottom.

$R''$ has no vertex, as depicted in Figs. 17(g) and (h), and hence the degree of $v$ is three in the original floorplan $R$. Then $v$ is $s$-missing, $w$-missing, or $e$-missing in $R$, as illustrated in Figs. 17(b), (c), and (d), respectively. Therefore, the next direction of the traversal is to the top or to the bottom; see Fig. 19(a). Hence, we only need one bit to represent the next direction.

**Case 2:** $e$ is traced from left to right.

If the degree of $v$ is two, then the next direction is always to the bottom, as depicted in Fig. 17(f). Otherwise, the
degree of \( v \) is three, \( v \) is n-missing, s-missing, or e-missing, as depicted in Figs. 17(a), (b), and (d), respectively. Therefore, the next direction is to the bottom or to the right. Hence, we need only one bit to represent the next direction; see Fig. 19(b).

**Case 3**: \( e \) is traced from bottom to top.

The degree of \( v \) is three or two. If the degree of \( v \) is three, then \( v \) is w-missing, e-missing, or n-missing, as illustrated in Figs. 20(a), (b), and (c), respectively. Note that, since \( v \) has an edge that goes down, \( v \) is not s-missing. If the degree of \( v \) is two, then \( v \) is the upper-left or upper-right vertex of the outer face. Figures 20(d) and (e) show illustrations of \( v \).

In this case, since we already have traversed from \( v \) to the bottom, it is sufficient to determine whether there is an edge from \( v \) to the left. If \( v \) has no leftward edge, that is, \( v \) is w-missing as in Fig. 20(a) or it is the upper-left vertex with degree two as in Fig. 20(d), then the next direction is always to the right. Otherwise, \( v \) has a left edge, and the next direction is to the left, as in Figs. 20(b) and (e), or to the right, as in Fig. 20(c). In this way, we have only two possibilities, and hence we can represent the next direction by using only one bit.

Now we estimate the number of bits used for this case. The number of bits for Case 3 is equal to the sum of the number of e-missing vertices, n-missing vertices, and the upper-right vertex. Thus by Eq. (2.3), we have

\[
\frac{n_E + n_N + 1}{2} = \frac{n - 4}{2} + 1 = \frac{n}{2} - 1.
\]

Recall that \( n_E \) and \( n_N \) are the numbers of e-missing and n-missing vertices, respectively.

**Case 4**: \( e \) is traced from right to left.

The degree of \( v \) is three or two. If the degree \( v \) is three, then \( v \) is w-missing, s-missing, or n-missing, as illustrated in Figs. 21(a), (b), and (c). Note that, since \( v \) has an edge to the left, \( v \) is not e-missing. If the degree \( v \) is two, then \( v \) is the upper-left vertex with degree two. Figure 21(d) illustrates \( v \) in this case.

In this case, we already have traversed the edges from \( v \) to the left and to the bottom. Hence it is sufficient to determine whether (1) there is an edge from \( v \) to the left, and (2) there is an edge from \( v \) to the bottom. If \( v \) is the upper-left vertex of the outer face, then we complete the traversal and we do not need to store the next direction. Otherwise, if there is no edge from \( v \) to the left, then the next direction is always to the top (Fig. 21(a)), and hence, again, we do not need to store the next direction. Otherwise, if there is an edge from \( v \) to the left, then \( v \) is s-missing as in Fig. 21(b) or n-missing as in Fig. 21(c). In both cases, the next direction is always to the left, and, yet again, we do not need to store the next direction. Therefore, Case 4 does not require any bits.

Based on the above case analyses, we can generate a code for any floorplan \( R \). We call this code the *improved DFS-based code* for \( R \).

**Theorem 5.2** ([26]). *One can encode a floorplan with \( f \) inner faces and \( B \) bottom edges by using \( 4f - B + 1 \) bits.*

**Proof.** We estimate the length of the code. For Cases 1 and 2, we use one bit to represent the next direction in each edge trace. Therefore we use at most \( m - B \) bits for the two cases in total, where \( m \) is the number of edges in \( R \). For Case 3, we use \( \frac{n}{2} - 1 \) bits, and for Case 4 we use no bits. Hence, by Eqs. (2.1) and (2.2), the total length of the code can be bounded by

\[
(m - B) + \left( \frac{n}{2} - 1 \right) = 3f + 1 - B + f = 4f - B + 1,
\]

which completes the proof.
As an example, Fig. 18(d) shows the traces of the depth-first traversal and the improved DFS-based code. The numbers of bits saved are indicated in parentheses. The improved DFS code for the floorplan is

```
00010101011101101100100101100100110011
```

The length of the improved DFS-based code for the floorplan in Fig. 18(a) is 38 bits, while the length of the DFS-based code for the same floorplan is 65 bits.

Furthermore, we can improve the code by further encoding the last part of the code. Let \( T \) be the number of U-active faces. Recall that an inner face \( F \) is U-active if \( \text{north}(F) = \emptyset \). For example, \( T = 2 \) holds for the floorplan in Fig. 18(a). If \( T = 1 \) holds, then we remove the U-active face. We repeat this process until the obtained floorplan holds \( T / 2 \geq 2 \). Let \( k \) be the number of removed faces, and let \( P \) be the obtained floorplan. The obtained floorplan \( P \) has \( f - k \) inner faces. To store this removing process, we insert a unary code for \( k + 1 \) into the prefix of the code. We note that \( k \) may be equal to zero. If \( f = k \) holds, then a floorplan can be encoded by using \( f + 1 \) bits. Otherwise, we obtain a floorplan \( P \) with \( T > 2 \).

Let \( F \) be the inner face of \( P \) that has the upper-right vertex of the outer face, and hence \( \text{north}(F) = \emptyset \) and \( \text{east}(F) = \emptyset \). We then have the following lemma for the suffix of an improved DFS-based code for \( P \).

**Lemma 5.3 ([26]).** Let \( S \) be the improved DFS-based code of any floorplan having at least two U-active faces. The six-bit suffix of \( S \) is always “010011” or “110011.”

**Proof.** We give a case analysis based on \( |\text{west}(F)| \).

**Case 1:** \( |\text{west}(F)| > 1 \)

The final directions of the traversal of \( P \) are right, down, up, right, down, up, then left \( T \) times, as in Fig. 22(a). Since all the bits for the edges on the north line segment of the outer face, excluding the rightmost such edge, are saved in \( S \), the six-bit suffix of \( S \) is “010011.”

**Case 2:** \( |\text{west}(F)| = 1 \)

Let \( v \) be the lower-left vertex of \( F \). We consider the following two subcases.

**Case 2a:** \( v \) is s-missing

The final directions of the traversal of \( P \) are right, down, up, right, down, up, left \( T \) times, as in Fig. 22(b). This is the same as Case 1, and the six-bit suffix is “010011.”

**Case 2b:** \( v \) is w-missing

The final directions are right, down, up, lefts \( x(x \geq 1) \) times, up, right, down, up, then left \( T \) times, as in Fig. 22(c). The bits for up followed by \( x \) lefts are saved in \( S \). Hence the six-bit suffix is “110011.”

By the above lemma, if we append one bit to distinguish Case 2a or Case 2b, then we can save the six-bit suffix of the improved DFS-based code for any floorplan that has two or more U-active faces. We thus have the following theorem.

**Theorem 5.4 ([26]).** One can encode any floorplan with \( f \) inner faces and \( B \) bottom edges by using \( 4f - B - 3 \) bits. Coding and decoding each take \( \Theta(f) \) time.

**Proof.** The length of the above code is

\[
 k + 1 + 4(f - k) - B + 1 - 6 + 1 = 4f - 3k - B - 3 \leq 4f - B - 3,
\]

which completes the proof.

We observe that, since \( B > 0 \) holds for any floorplan, the code requires at most \( 4f - 4 \) bits.
6. Representation with Efficient Query Supports Based on the Dual Graph

In the previous sections, we focused on only the compactness of the codes for floorplans. In this section, we will focus not only on the compactness but also on the functionality of the codes. The code discussed in this section uses $5f + o(n)$ bits, but it supports queries about both adjacency and degree in constant time [34]. As we mentioned in Section 2.2, if we represent a given graph $G$ with $n$ vertices and $m$ edges by using the adjacency matrix representation, then, in $\Theta(1)$ time, we can determine whether two particular vertices are adjacent, and in $\Theta(n)$ time, we can compute the degree of a given vertex. Otherwise, if we represent $G$ by using the adjacency list representation, then we can determine in $\Theta(\Delta)$ time whether two vertices are adjacent, where $\Delta$ is the maximum degree of $G$, and in $\Theta(\Delta)$ time, we can compute the degree of a given vertex. However, the code in [34] represents a floorplan by using compact code that supports adjacency and degree queries in constant time. Note that the maximum degree of the vertices of a floorplan $R$ is three, although the maximum degree of faces, namely the number of neighbour faces, is not bounded by a constant.

6.1 Code for Floorplans

In this subsection, given a floorplan $R$ with $f$ inner faces, we design a binary code $S_R$ for $R$ such that (1) $S_R$ can be decoded to reconstruct $R$, and (2) the length of $S_R$ is at most $5f + 9$ bits. In Sections 6.2, 6.3, and 6.4, by using auxiliary tables $S_A$ of length $o(f)$ bits, we explain how to query $R$ in $\Theta(1)$ time. The details of $S_A$ are shown in Section 6.2. The idea of our coding scheme is shown in Fig. 23.

We first provide some definitions. Given a floorplan $R$, and by rotating it clockwise by 0, 90, 180, or 270 degrees, one can have four floorplans. Without loss of generality, we can assume that $R$ is the floorplan that has the most $s$-missing vertices. Otherwise, we can rotate $R$ until it has the most $s$-missing vertices, and store the amount of rotation in two bits. This rotation information can then be used prior to each query. The following lemma gives a property of the number of $s$-missing vertices in this $R$.

**Lemma 6.1** ([34]). Let $n_s$ be the number of $s$-missing vertices. Then $n_s \geq \frac{n-4}{4}$ holds, where $n$ is the number of vertices of $R$.

**Proof.** The four vertices of the outer face of $R$ have degree two, and all other vertices have degree three. Each vertex with degree three is either $w$-, $e$-, $n$-, or $s$-missing. Since $n_{s}$ is the largest, $n_{s} \geq \frac{n-4}{4}$ holds. \qed

Given a floorplan $R$, we compute the planar dual $D_R$ of $R$ as follows. First, by extending the north and south line segments of the outer face, we divide the outer face of $R$ into four faces, which we will call the west, east, north, and south outer faces. We put a vertex in each face of $R$, and we connect two vertices by an edge if the corresponding two faces are adjacent; see Fig. 23(b). Each of the divided four outer faces is also replaced with a vertex. Next, we classify
the edges of $D_R$ into two subsets, as follows. Each edge of $D_R$ corresponds to some adjacency of two faces of $R$, and it is either ns-adjacent or ew-adjacent. Let $E_{NS}$ be the set of edges of $D_R$ corresponding to the ns-adjacencies, and let $E_{EW}$ be the set of edges corresponding to the ew-adjacencies; see Fig. 23(c), where the thick lines are the edges in $E_{NS}$ and the thin lines are the edges in $E_{EW}$. Let $D_{NS}$ be the subgraph of $D_R$ induced by the edges in $E_{NS}$, and let $D_{EW}$ be the subgraph of $D_R$ induced by the edges in $E_{EW}$. Next, we further classify the edges in $E_{NS}$ into two subsets. For each face $F$ of $R$, let $p(F)$ be the westmost inner face in $north(F)$. We call $p(F)$ the parent face of $F$. For convenience, we suppose that the north outer face $f_N$ has no parent face. Let $E^T_{NS}$ be the set of edges each of which corresponds to an adjacency between face $F$ and $p(F)$. Clearly, $E^T_{NS}$ induces a spanning tree of $D_R$, which we will call $T_{NS}$. An example is shown in Fig. 23(d), in which $T_{NS}$ is indicated by thick lines, while the edges in $E_{NS} \setminus E^T_{NS}$ are indicated by dashed lines. We will construct the code $S_1$ from $T_{NS}$, then construct the code $S_2$ from $T_{NS}$ and $D_{EW}$, and finally construct the code $S_R = S_1 + S_2$ for $R$.

We construct $S_1$ from $T_{NS}$, as follows. (It is identical to the code for an ordered tree that was discussed in Section 1.) Starting at the vertex corresponding to the north outer face, we traverse $T_{NS}$ in a depth-first manner. If we go down an edge, then we code it with $\langle$ and if we go up an edge, we code it with $\rangle$. Let $S_1$ be the resulting code. We define $S_1 = (S'_1)$, that is, the code obtained from $S_1$ by appending (at the head and) at the end. (Actually, we encode each (as 0 and each ) as 1.) Figure 24 shows a code $S_1$ of the floorplan in Fig. 23. The $i$-th (and its matching parenthesis) in $S_1$ correspond to the $i$-th vertex of $T_{NS}$ in the preorder.

We now construct $S_2$, as shown in the following steps. We first construct $S'_2$ from $S_1$. We denote by $F_i$ the inner face corresponding to the $i$-th vertex of $T_{NS}$ in the preorder. We replace the $i$-th (with $\{west(F_i)\}$ consecutive $]$s, and we replace its matching parenthesis $]$ with $\{east(F_i)\}$ consecutive $[$s. We note that, since $west(F_i) = west(F_{i+1}) = west(F_1) = \emptyset$ and $east(F_i) = east(F_{i+1}) = east(F_1) = \emptyset$ hold, they have no corresponding code. We also note that every inner face $F_i$ of $R$ satisfies $\{west(F_i)\} \geq 1$ and $\{east(F_i)\} \geq 1$. The $\{west(F_i)\}$ consecutive $]$s in $S'_2$ correspond to the $\{west(F_i)\}$ ew-adjacencies between the faces in $west(F_i)$ and $F_i$. Also, the $k$-th ] among the $\{west(F_i)\}$ consecutive $]$s corresponds to the $k$-th ew-adjacency of $F_i$ in counterclockwise order. Similarly, the $\{east(F_i)\}$ consecutive $[$s correspond to the $\{east(F_i)\}$ ew-adjacencies between the faces in $east(F_i)$ and $F_i$. Also, the $k$-th [ among $\{east(F_i)\}$ consecutive $[$s corresponds to the $k$-th ew-adjacency of $F_i$ in counterclockwise order. Note that since $T_{NS}$ is a tree, $S_2$ has a nested structure. (We will explain the reason for this in Section 6.3.) Next, we construct $S_2$ from $S'_2$. We replace each of the $\{west(F_i)\}$ consecutive $]$s with $\{(west(F_i)) - 1\}$ consecutive $0$s, followed by 1. That is, $\{west(F_i)\}$ is represented by a unary code. Similarly, we replace each of the $\{east(F_i)\}$ consecutive $[$s with $\{(east(F_i)) - 1\}$ consecutive $0$s, followed by 1. An example is shown in Fig. 24. Each 1 in $S_2$ represents the border of either $west(F_i)$ or $east(F_i)$. Note that we store no information about the edges in $E_{NS} \setminus E^T_{NS}$ (however, we will show that such information can be reconstructed).

Now we estimate the length of $S_R = S_1 + S_2$. Let $n$ be the number of vertices of $R$, let $m$ be the number of edges of $R$, and let $j$ be the number of inner faces of $R$. Then $D_R$ has $m + 4$ edges, because we extend the north and south line segments of the outer face of $R$. First we estimate the length of $S_1$. The code $S'_1$ stores the pair of ( and ) for each edge of $T_{NS}$, then by adding two more bits we obtain $S_1$. Thus $|S_1| = 2|E^T_{NS}| + 2$. We estimate the length of $S_2$ as follows. Since the pair of [ and ] is stored for each edge corresponding to some ew-adjacency, $|S_2| = 2|E_{EW}| = 2((m + 4) - |E_{NS}|)$. Therefore,

$$|S_R| = |S_1| + |S_2|$$

$$= (2|E^T_{NS}| + 2) + (2m + 8 - 2|E_{NS}|)$$

$$= 2m + 10 - 2|E_{NS} \setminus E^T_{NS}|.$$

Fig. 24. The code $S_R = S_1 + S_2$ for the floorplan in Fig. 23.
and we have the following lemma.

**Lemma 6.2** ([34]). \( |E_{NS} \setminus E_{ENS}^S| = ns + 2 \), where ns is the number of s-missing vertices of R.

**Proof.** We show that there is a one-to-one mapping between the s-missing vertices and edges in \( E_{NS} \setminus E_{ENS}^S \). We assume that face \( F_a \) is a north neighbour of a face \( F_b \) and that \( F_a \neq p(F_b) \). Let \( a \) and \( b \) be the vertices corresponding to \( F_a \) and \( F_b \), respectively. Then we can assign the lower-left vertex \( x \) of the face \( F_a \), which is s-missing, to the edge \( (a, b) \). Since the definition of \( E_{ENS}^S \) implies that each vertex has at most one downward edge in \( E_{NS} \setminus E_{ENS}^S \), there is no duplication in these assignments. Since the division of the outer face has created two s-missing vertices in \( R \), the claim holds.

By Lemma 6.2,

\[
|S_R| = 2m + 10 - 2|E_{NS} \setminus E_{ENS}^S| \\
= 2m + 10 - 2(ns + 2).
\]

By Lemma 6.1,

\[
|S_R| \leq 2m + 10 - 2\left(\frac{n - 4}{4} + 2\right) \\
= 2m - \frac{n}{2} + 8.
\]

By Eqs. 2.1 and 2.2, \( |S_R| \leq 5f + 9 \) holds. In Section 6.5, we will show that \( S_R \) can be decoded to reconstruct \( R \).

### 6.2 The Basic Queries

This subsection considers basic queries, which are necessary to support adjacency and degree queries about the faces of floorplans, which will be considered in Sections 6.3 and 6.4. We first briefly survey some basic queries for binary codes; then, by applying these queries to the code \( S_R \) of a given floorplan \( R \), we will be able to support some basic queries on \( R \).

Let \( S[i, j] \) be the subcode of a code \( S \) from position \( i \) to position \( j \). We denote \( S[i, j] = S[i] \). Let \( S_i \) be the code in Section 6.1 consisting of \( \langle \rangle \) and \( \langle \rangle \); let \( \text{rank}(S_i, i, j) \) denote the number of \( \langle \rangle \) up to and including the position \( i \) in \( S_i \), and let \( \text{select}(S_i, k, j) \) denote the position of the \( k \)-th \( \langle \rangle \) in \( S_i \). We note that if \( j = \text{select}(S_i, k, \langle \rangle) \), then \( k = \text{rank}(S_i, j, \langle \rangle) \). In a similar way, we define \( \text{rank}(S_i, i, j) \) and \( \text{select}(S_i, k, j) \). For example, if \( S_i = \langle (\langle \rangle) \rangle \langle (\langle \rangle) \rangle \rangle \langle \rangle \langle \rangle \rangle \langle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle \langle \rangle \rangle \rangle
\]

In much of the literature, this operation is referred to as \text{findclose} and \text{findopen}. See, e.g., [20].
Similarly, let \( F_{WN} \) be the northernmost face in \( west(F_i) \). Then, we write \( F_{WN}(i) = j. \) \( F_{WN} \) is the northernmost (\( = N \)) face among the west (\( = W \)) neighbours of \( F_i \). \( F_{WN} \) is calculated as follows:

\[
F_{WN}(i) = \text{rank}(S_1, \text{match}(S_1, \text{rank}(S_2, p_2, 1) + 5), ()).
\]

Similarly, let \( F_{WS} \) be the southernmost face in \( west(F_i) \). Then,

\[
F_{WS}(i) = \text{rank}(S_1, \text{match}(S_1, \text{rank}(S_2, \text{match}(S_2, a - 4, 1)), 1) + 5), ()).
\]

Similarly, let \( R_i \) be the subcode in \( S_2 \) corresponding to \( east(F_i) \). We can find \( R_i \) in \( S_2 \) as follows. Assume \( \text{select}(S_2, i, 1) = a \) and \( \text{match}(S_1, a) = b \). Then \( R_i = S_2[\text{select}(S_1, a - 5, 1) + 1, \text{select}(S_1, a - 4, 1)] \) holds. \( R_i \) corresponds to the \( |\text{east}(F_i)| = |R_i| \) of ew-adjacencies between faces in \( east(F_i) \) and \( F_i \) corresponding to the \( i \)-th vertex. Also the \( k \)-th \( [ \) in \( R_i \) corresponds to the \( k \)-th ew-adjacency of \( F_i \) in counterclockwise order. Let \( F_{EN} \) and \( F_{ES} \) be the northernmost and southernmost faces in \( east(F_i) \), respectively. Then, similarly,

\[
F_{EN}(i) = \text{rank}(S_1, \text{rank}(S_2, \text{match}(S_2, \text{select}(S_1, a, 1)), 1) + 5), ()
\]

\[
F_{ES}(i) = \text{rank}(S_1, \text{rank}(S_2, \text{match}(S_2, \text{select}(S_1, a - 5, 1)), 1) + 5), ()
\]

Lemma 6.4 ([34]). Given a balanced code \( S_R = S_1 + S_2 \) in Section 6.1, using auxiliary tables \( S_A \) of \( o(f) \) bits, one can compute \( L_i, R_i, F_{WN}(i), F_{WS}(i), F_{EN}(i), \) and \( F_{ES}(i) \) in \( O(1) \) time. One can construct \( S_A \) in \( O(f) \) time.

Proof. It is trivial by Lemma 6.3 and the above observations. \( \square \)

Let \( v \) be a lower-left vertex of an inner face \( F_i \) corresponding to the \( j \)-th vertex of \( T_N \) in the preorder. Then, \( v \) is either \( s \)-missing or \( w \)-missing. We can compute which direction of \( v \) is missing, as follows: if \( F_{ES}(F_{WS}(i)) = i \), then \( v \) is \( s \)-missing, otherwise, \( v \) is \( w \)-missing, and we have the following lemma.

Lemma 6.5 ([34]). For each of four vertices of an inner face, one can compute which direction of the vertex is missing in \( O(1) \) time.

6.3 Adjacency Query

Now we can explain how an adjacency query on the faces can be computed in \( O(1) \) time. We can compute such a query with the help of the auxiliary tables \( S_A \) of \( o(f) \) bits. Given two faces \( F_i \) and \( F_j \) of a floorplan \( R \), we consider the following four adjacency queries.

\( (\text{wa}) \) \( F_j \in west(F_i) \): is \( F_j \) a west neighbour of \( F_i \)? (See Fig. 25(a).)

\( (\text{ea}) \) \( F_j \in east(F_i) \): is \( F_j \) an east neighbour of \( F_i \)? (See Fig. 25(b).)

\( (\text{na}) \) \( F_j \in north(F_i) \): is \( F_j \) a north neighbour of \( F_i \)? (See Fig. 25(c).)

\( (\text{sa}) \) \( F_j \in south(F_i) \): is \( F_j \) a south neighbour of \( F_i \)? (See Fig. 25(d).)

Let \( L_i \) and \( L_j \) be the subcodes in \( S_2 \) corresponding to \( west(F_i) \) and \( west(F_j) \), respectively. Let \( R_i \) and \( R_j \) be the subcodes in \( S_2 \) corresponding to \( east(F_i) \) and \( east(F_j) \), respectively. We denote by \( D_R \) be the planar dual of \( R \). Now we explain how to make these four adjacency queries.

Query \( (\text{wa}) \)

We note that \( F_j \in west(F_i) \) if and only if some \( [ \in R_i \) matches some \( ] \) in \( L_i \). Such a pair can be computed as follows. Assume that \( R_i = S_2[a, b] \) and \( L_i = S_2[c, d] \). Without loss of generality, we can assume that \( a < b < c < d \). We then have the following three cases.

Case 1: \( \text{match}(S_2, d) > b \).

No \( [ \) in \( L_i \) matches any \( ] \) in \( R_i \). Hence \( F_j \notin west(F_i) \).

Case 2: \( a \leq \text{match}(S_2, d) \leq b \).

In this case, \( S_2[d] \) matches some \( ] \) in \( R_i \). Hence \( F_j \in west(F_i) \).

Case 3: \( \text{match}(S_2, d) < a \).

We have two subcases.

Case 3a: \( \text{match}(S_2, a) < b \).

No \( [ \) in \( R_j \) matches any \( ] \) in \( L_i \). Thus \( F_j \notin west(F_i) \).

Case 3b: \( c \leq \text{match}(S_2, a) < d \).

In this case, \( S_2[a] \) matches some \( ] \) in \( L_i \). Hence \( F_j \in west(F_i) \).

By the above case analysis, we can compute query \( (\text{wa}) \) in \( O(1) \) time.
Query (ea)

The computation for query (ea) is symmetric to query (wa), and so we omit the details.

Query (na)

If $F_j$ is the parent face of $F_i$, then $F_j \in \text{north}(F_i)$. Otherwise, we have to compute whether $F_j$ is a north neighbour of $F_i$ by the edge in $\mathcal{EN}_S \setminus \mathcal{EN}_S^i$ is stored (directly) in $S_1$ or $S_2$. However, we can compute the adjacency query (na) in the following way.

First, we show that the nested structure of $S_2$ corresponds to an inclusion structure of the regions of the planar dual $D_R$. We need some definitions. Let $\{v_1, v_2, \ldots, v_{f+1}\}$ be the set of vertices of $D_R$. The root $v_1$ of $T_{NS}$ corresponds to the north (outer) face. For each edge $e = (v_j, v_k) \in \mathcal{EW}$, we assign the following region $R(v_j, v_k)$ of $D_R$ as follows. Let $P_i$ be the path $v_i$ to $v_j$, and let $P_k$ be the path $v_k$ to $v_j$. Then $R(v_j, v_k)$ is the region enclosed by the two paths $P_i, P_k$, and edge $e$. No region $R(v_j, v_k)$ can include only a proper part of the other. (Since, otherwise, if we assume that $R(v_j, v_k)$ includes a proper part of $R(v_j, v_k)$, then now there is an edge $(v_j, v_k) \in \mathcal{EW}$ such that $v_j \in R(v_j, v_k)$ and $v_k \notin R(v_j, v_k)$; however, then edge $(v_j, v_k)$ intersects some edge on the boundary of $R(v_j, v_k)$, which is a contradiction.) By the planarity of $D_R$, we have the following lemma.

Lemma 6.6 ([34]). $R(v_j, v_k)$ properly includes $R(v_j, v_k)$ if and only if the pair of $[\ ]$ and $[\ ]$ for edge $(v_j, v_k)$ encloses the pair of $[\ ]$ and $[\ ]$ for edge $(v_j, v_k)$.

---

Fig. 25. Illustration for how to compute the four adjacency queries.
Lemma 6.7 \[\text{consecutive}\]

Proof. By the planarity of \(D_R\).

Now let us consider face \(F_j\), which is a north neighbour of \(F_i\), but \(F_j \neq p(F_i)\). Assume \(k = F_{EN}(i)\), and that its corresponding vertex is \(v_k\). Let \(v\) be the upper-right vertex of \(F_i\). If \(v\) is n-missing, then we assume \(l = F_{WS}(j)\). (See Fig. 26(a).) Otherwise, we assume \(l = F_{ES}(j)\); see Fig. 26(b). Let \(v_l\) be the \(l\)-th vertex of \(T_{NS}\) in the preorder. Each of the two edges \((v_l, v_k)\) and \((v_j, v_l)\) represents some ew-adjacency. Let \(R(v_l, v_k)\) be the region obtained from region \(R(v_j, v_l)\) by eliminating the face with edge \((v_j, v_l)\) on its contour. Then the pair of \([\text{and}]\) for edge \((v_l, v_k)\) immediately encloses the pair of \([\text{and}]\) for edge \((v_j, v_l)\) if and only if the region \(R(v_l, v_k)\) has edge \((v_j, v_l)\) on the contour of \(R(v_l, v_k)\). We then have the following lemma.

Lemma 6.7 ([34]). Suppose that \(F_j\) is not the parent face of \(F_i\). \(F_j\) is a north neighbour of \(F_i\) if and only if the pair of \([\text{and}]\) for edge \((v_l, v_k)\) immediately encloses the pair of \([\text{and}]\) for edge \((v_j, v_l)\). (See Fig. 26.)

Proof. \((\Rightarrow)\) Since \(F_j\) is a north neighbour of \(F_i\), region \(R(v_l, v_k)\) includes region \(R(v_j, v_l)\); see Fig. 26. Then the pair of \([\text{and}]\) for \((v_l, v_k)\) encloses the pair of \([\text{and}]\) for \((v_j, v_l)\). In addition, when the face having edge \((v_l, v_k)\) on its contour is eliminated from \(R(v_l, v_k)\), \((v_j, v_l)\) appears on the contour of the resulting region. Thus the pair of \([\text{and}]\) for \((v_l, v_k)\) immediately encloses the pair of \([\text{and}]\) for \((v_j, v_l)\).

\((\Leftarrow)\) Since the pair of \([\text{and}]\) for \((v_l, v_k)\) immediately encloses the pair of \([\text{and}]\) for \((v_j, v_l)\), when the face having edge \((v_l, v_k)\) on its contour is eliminated from \(R(v_l, v_k)\), \((v_j, v_l)\) appears on the contour. Thus \(F_j\) is a north neighbour of \(F_i\), as shown in Fig. 26. Note that each of the two edges \((v_l, v_k)\) and \((v_j, v_l)\) corresponds to some ew-adjacency.

By Lemma 6.7, we can compute the query (na) in the following way.

Case 1: \(F_j\) is the parent face of \(F_i\).

\(F_j \in \text{north}(F_i)\). By using the enclose operation in \(S_1\), we can compute whether \(F_j\) is the parent face of \(F_i\) in \(O(1)\) time.

Case 2: Otherwise.

By Lemma 6.7, the pair of \([\text{and}]\) for edge \((v_l, v_k)\) immediately encloses the pair of \([\text{and}]\) for edge \((v_j, v_l)\) if and only if \(F_j \in \text{north}(F_i)\). Let \(v\) be the upper-right vertex of \(F_i\). We have the following two subcases. Assume that \(a = \text{match}(S_1, \text{select}(S_1, i, ()))\), that is, \(a\) is the position of \(v\) in \(S_1\).

Case 2a: \(v\) is n-missing. (See Fig. 26(a).)

In this case, \(l = F_{WS}(j)\). Assume \(b = \text{select}(S_1, j, ())\). Then

\[
\text{enclose}(S_2, \text{select}(S_2, b - 4, 1)) = \text{select}(S_2, a - 4, 1)
\]

if and only if \(F_j \in \text{north}(F_i)\), since \(\text{select}(S_2, b - 4, 1)\) in the above equation indicates the last \(\text{select}(b)\) among \(\text{select}(j)\) consecutive \(\text{select}(b)\). Note that the indicated \(\text{select}(b)\) above corresponds to edge \((v_j, v_l)\).

Case 2b: \(v\) is e-missing. (See Fig. 26(b).)

In this case, \(l = F_{ES}(j)\). Similar to Case 2a, assume \(c = \text{match}(S_1, \text{select}(S_1, j, ()))\). Then

\[
\text{enclose}(S_2, \text{select}(S_2, c - 5, 1) + 1) = \text{select}(S_2, a - 4, 1)
\]

if and only if \(F_j \in \text{north}(F_i)\).

Therefore we can compute query \(\text{(na)}\) in \(O(1)\) time.

Query (sa)

Finally we consider the query (sa). \(F_i\) is a north neighbour of \(F_j\) if and only if \(F_j\) is a south neighbour of \(F_i\). Thus we can compute the query (sa) by using the query (na).

Now, using the queries (wa), (ea), (na), and (sa), we have the following theorem.

Theorem 6.8 ([34]). Given the code \(S = S_R + S_A\) for any floorplan \(R\) and two inner faces \(F_i\) and \(F_j\) of \(R\), one can
compute whether \( F_i \) and \( F_j \) are adjacent in \( \Theta(1) \) time.

6.4 Degree Query

In this section we explain how to compute the number of neighbour faces for a given inner face. We can compute such a query in \( \Theta(1) \) time with the help of the auxiliary tables \( S_\delta \) of \( o(f) \) bits. We consider the following four degree queries (wd), (ed), (nd), and (sd) for each direction.

**Case 1**

We have the following theorem.

\[
\text{(wd): compute } |\text{west}(F_i)|
\]

In Section 6.2, we explained how to find \( L_i \) in \( S_2 \). Since \( |\text{west}(F_i)| = |L_i| \), we can compute the query \( \text{wd} \) in \( \Theta(1) \) time.

**Case 2**

We have the following theorem.

\[
\text{(ed): compute } |\text{east}(F_i)|
\]

Since \( |\text{east}(F_i)| = |R_i| \), we can compute the query \( \text{ed} \) in \( \Theta(1) \) time.

**Case 3**

We have the following theorem.

\[
\text{(nd): compute } |\text{north}(F_i)|
\]

Let \( v_i \) be the vertex corresponding to \( F_i \), and let \( F_k \) be the northernmost east neighbour of \( F_i \), where \( k = F_{E_\delta}(i) \). Let \( v_k \) be its corresponding vertex. We denote by \( \delta_1, \delta_2, \ldots, \delta_{|\text{north}(F_i)|} \) the north neighbours of \( F_i \) in clockwise order. Let \( v_1^\delta, v_2^\delta, \ldots, v_{|\text{north}(F_i)|}^\delta \) be the vertices corresponding to \( \delta_1, \delta_2, \ldots, \delta_{|\text{north}(F_i)|} \), respectively. Note that \( \delta_1 \) is the parent face of \( F_i \). Let \( v \) be the upper-right vertex of \( F_i \). We have the following two cases.

**Case 1:** \( v \) is \( m \)-missing. (See Fig. 26(a).)

Assume that \([ \text{and } ]\) in \( S_2 \) for edge \((v_j, v_k)\) are at positions \( a \) and \( b \), respectively, in \( S_2 \). Note that the values of \( a \) and \( b \) can be computed in \( \Theta(1) \) time by using basic queries. Then the pairs immediately enclosed by the pair of \( S_2[a] \) and \( S_2[b] \) are only pairs of \([ \text{and } ]\) for edges \((v_1^\delta, v_2^\delta), (v_2^\delta, v_3^\delta), \ldots, (v_{|\text{north}(F_i)|-1}^\delta, v_{|\text{north}(F_i)|}^\delta)\). Thus \( |\text{north}(F_i)| = (\text{wrapped}(S_2, a))/2 \).

**Case 2:** \( v \) is \( e \)-missing. (See Fig. 26(b).)

Similar to Case 1, assume that \([ \text{and } ]\) in \( S_2 \) for edge \((v_j, v_k)\) are at positions \( a \) and \( b \), respectively, in \( S_2 \). Then the pairs immediately enclosed by the pair of \( S_2[a] \) and \( S_2[b] \) are only pairs of \([ \text{and } ]\) for edges \((v_1^\delta, v_2^\delta), (v_2^\delta, v_3^\delta), \ldots, (v_{|\text{north}(F_i)|-1}^\delta, v_{|\text{north}(F_i)|}^\delta), (v_{|\text{north}(F_i)|}^\delta, v_k)\). Thus \( |\text{north}(F_i)| = \text{wrapped}(S_2, a)/2 \).

Therefore we can compute query \( \text{nd} \) in \( \Theta(1) \) time.

**Case 4**

We have the following theorem.

\[
\text{(sd): compute } |\text{south}(F_i)|
\]

Let \( v \) be the lower-left vertex of \( F_i \). We have the following two cases.

**Case 1:** \( v \) is \( w \)-missing.

In this case, a face \( F \) is a south neighbour of \( F_i \) if and only if its parent face is \( F_i \). Thus we need to compute the number of faces with the parent face \( F_i \). Hence \( |\text{south}(F_i)| = |\text{wrapped}(S_1, \text{select}(S_1, i, \langle \rangle))/2 \). Therefore we can compute \( \text{south}(F_i) \) in \( \Theta(1) \) time.

**Case 2:** \( v \) is \( s \)-missing.

In this case, there is exactly one south neighbour of \( F_i \) such that its parent face is not \( F_i \). Such a face is the westernmost south neighbour of \( F_i \). For the other faces, the condition is similar to Case 1. Thus \( |\text{south}(F_i)| = 1 + \text{wrapped}(S_1, \text{select}(S_1, i, \langle \rangle))/2 \).

Therefore we can compute the number of neighbour faces of \( F_i \) in the designated direction in \( \Theta(1) \) time, and we have the following theorem.

**Theorem 6.9** ([34]). Given the code \( S = S_R + S_\delta \) for a floorplan \( R \), one can compute the degree of \( F_i \) in \( \Theta(1) \) time.

**Proof.** The degree of \( F_i \) is equal to \( |\text{west}(F_i)| + |\text{east}(F_i)| + |\text{north}(F_i)| + |\text{south}(F_i)| \).

6.5 Reconstruction of \( R \)

Given the code \( S = S_R + S_\delta \) for a floorplan \( R \), we can reconstruct \( R \) by using the queries presented in the previous sections. We have the following theorem.

**Theorem 6.10** ([34]). Given the code \( S_R \) of any floorplan \( R \), one can construct auxiliary tables \( S_\delta \) of \( o(f) \) bits in \( \Theta(n) \) time. Then one can reconstruct \( R \) from \( S \) in \( \Theta(f) \) time.

**Proof.** Proof by induction. If we know about all the west and north neighbours for each \( 1, 2, \ldots, (i-1) \)-th face, we can also compute the west and north neighbours for the \( i \)-th face in \( \Theta(|\text{north}(F_i)| + |\text{west}(F_i)|) \) time by using the queries presented. Thus the running time of the algorithm is \( \Theta(f) \) in total.
7. Representation for Mosaic Floorplans

In the previous sections, we gave compact codes for floorplans. In this section, we give a code for a “mosaic floorplan,” which is a variation that is also used for VLSI floorplanning [12]. In this section, we present an optimal code [12] for a mosaic floorplan. The code for any mosaic floorplan with \( f \) inner faces takes \( 3f - 3 \) bits, while the representation by Sakanushi et al. [27] uses \( 4f \) bits. See [12] for a survey of the representations of mosaic floorplans. Independently, Takahashi [30] also proposed a \((3n - 4)\)-bit code for a mosaic floorplan. The two codes are essentially similar; hence, in this paper, we will explain only He’s code [12].

7.1 Mosaic Floorplans

A mosaic floorplan is a partition (dissection) of a rectangle into smaller rectangles with vertical and horizontal line segments such that no four rectangles meet at the same point. An isomorphism of a mosaic floorplan is formally defined by using the horizontal-constraint graph of the line segments (see Fig. 27(a)) and the vertical-constraint graph of line segments (see Fig. 27(b)). The horizontal-constraint graph of line segments represents the horizontal relationship between the maximal vertical line segments. Similarly, the vertical-constraint graph of line segments represents the vertical relationship between the maximal horizontal line segments. Their formal definitions will be given below.

Let \( R \) be a mosaic floorplan. The horizontal-constraint graph \( D_H(R) = (V_H, A_H) \) of the line segments of \( R \) is a directed graph (see Fig. 27(a)), where \( V_H \) is a vertex set and \( A_H \) is a directed edge set. Each vertex in \( V_H \) corresponds to a maximal vertical line segment. For two vertices \( u, v \in V_H \), there is a directed edge from \( u \) to \( v \) if and only if there is an inner face \( F \) such that the west line segment of \( F \) is contained in the vertical line segment corresponding to \( u \), and the east line segment of \( F \) is contained in the vertical line segment corresponding to \( v \). Intuitively, an inner face corresponds to a directed edge in \( A_H \). The vertical-constraint graph \( D_V(R) = (V_V, A_V) \) of line segments of \( R \) is a directed graph (see Fig. 27(b)), where \( V_V \) is a vertex set and \( A_V \) is a directed edge set. Each vertex in \( V_V \) corresponds to a maximal horizontal line segment. For two vertices \( u, v \in V_V \), there is a directed edge from \( u \) to \( v \) if and only if there is an inner face \( F \) such that the south line segment of \( F \) is contained in the horizontal line segment corresponding to \( u \), and the north line segment of \( F \) is contained in the horizontal line segment corresponding to \( v \). Intuitively, an inner face corresponds to a directed edge in \( A_V \). Two mosaic floorplans are isomorphic if and only if their line segments have identical horizontal-constraint graphs and vertical-constraint graphs. For example, the three floorplans in Fig. 4 are isomorphic as mosaic floorplans. However, the two floorplans in Figs. 4(a) and (c) are non-isomorphic floorplans. Intuitively, two mosaic floorplans are isomorphic if and only if they can be converted into each other by sliding their horizontal and vertical line segments.

From the definition of the isomorphism of mosaic floorplans, we note that two distinct (normal) floorplans may be isomorphic as mosaic floorplans. Among such floorplans, we uniquely define the “canoncal” floorplan. A floorplan is canonical if any n-missing vertex appears to the left of any s-missing vertex on any horizontal line segment, and any w-missing vertex appears above any e-missing vertex on any vertical line segment. For example, the mosaic floorplan in Fig. 4(c) is a canonical floorplan, but those in Figs. 4(a) and (b) are not. We now have the following lemma.

Lemma 7.1. The canonical floorplan is unique among floorplans such that they are non-isomorphic as floorplans, but isomorphic as mosaic floorplans.

Proof. We will prove by assuming otherwise and arriving at a contradiction. Let \( C_1 \) and \( C_2 \) be two distinct canonical floorplans such that they are isomorphic as mosaic floorplans. Then, since \( C_1 \) and \( C_2 \) are isomorphic as mosaic floorplans, \( C_1 \) can be converted to \( C_2 \) by sliding horizontal and vertical line segments. Hence, one of two floorplans does not satisfy the condition of a canonical floorplan, which is a contradiction. \( \square \)
By Lemma 7.1 and the definition of canonical and mosaic floorplans, we observe that there is a natural one-to-one correspondence between canonical floorplans and mosaic floorplans with the same number of inner faces.

### 7.2 Information-Theoretic Lower Bounds for Mosaic Floorplans

Ackerman et al. [1] showed a bijection between mosaic floorplans with \( f \) inner faces and Baxter permutations on \( \{ f \} = \{ 1, 2, \ldots , f \} \). This can be regarded as a bijection between canonical floorplans with \( f \) inner faces and Baxter permutations on \( \{ f \} \). A Baxter permutation on \( \{ f \} \) is a permutation \( \pi = (\sigma_1, \sigma_2, \ldots , \sigma_f) \) for which there are no four indexes \( 1 \leq i < j < k < l \leq f \) such that (1) \( \sigma_i < \sigma_j + 1 = \sigma_l < \sigma_k \), or (2) \( \sigma_j < \sigma_l + 1 = \sigma_i < \sigma_k \). See [1] for further details. Let \( B(\pi) \) be the number of Baxter permutations on \( \{ f \} \). \( B(\pi) \) is called the \( f \)-th Baxter number, and it is defined as follows:

\[
B(\pi) = \frac{\sum_{r=0}^{f-1} \binom{f+1}{r} \binom{f+1}{r+1} \binom{f+1}{r+2}}{\binom{f+1}{1} \binom{f+1}{2}}.
\]

\( B(\pi) \) is denoted as \( B(\pi) = O(8^f / f^4) \) [29]. Hence the information-theoretic lower bounds of Baxter permutations on \( \{ f \} \) and mosaic floorplans with \( f \) inner faces are \( \log_2 B(\pi) = 3n - o(f) \) bits. Hence, the two codes in [12] and [30] of length \( 3n - O(1) \) bits are asymptotically optimal.

The coding and decoding algorithms for these two codes require linear time. The bijection in [1] gives a linear-time transformation from mosaic floorplans to Baxter permutations, and vice versa. Therefore, these two codes are also asymptotically optimal codes of a Baxter permutation, and their coding and decoding can be done in linear time.

### 7.3 Optimal Code for Mosaic Floorplans

In this section, we give a code for a canonical floorplan. As mentioned in Section 7.1, since canonical floorplans correspond to mosaic floorplans, the code can be regarded as a code for mosaic floorplans. The code is based on the staircase that was defined in Section 4.1.

Let \( C \) be a canonical floorplan with \( f \) inner faces, and let \( SS = (C_1, C_{f-1}, \ldots , C_f) \) be the staircase sequence of \( C \). Recall that \( C_i, 1 \leq i \leq f - 1 \), is obtained from \( C_{i+1} \) by deleting the top-ready inner face \( F_{i+1} \). \( C_i \) is the sub-floorplan induced by the inner faces \( F_1, F_2, \ldots , F_i \). See Fig. 28 for an example. The following lemma holds.

**Lemma 7.2** ([12]). \( C_i \) for each \( i = 1, 2, \ldots , f \) has exactly one top-ready inner face, denoted by \( F_i \), and \( F_i \) is adjacent to \( F_{i+1} \). If the lower-left vertex \( v \) of \( F_i \) is w-missing in \( C_i \), then \( F_{i+1} \) is the easternmost south neighbour of \( F_i \). Otherwise, \( v \) is s-missing in \( C_i \), then \( F_{i+1} \) is the northernmost west neighbour of \( F_i \).

**Proof.** Proof by induction. Clearly, in \( C_1 \), the inner face with the upper-right vertex of the outer face is the unique top-ready inner face \( F_1 \). Let \( F_0 \) and \( F_1 \) be the northernmost west neighbour of \( F_1 \) and the easternmost south neighbour of \( F_1 \). We show that either \( F_0 \) or \( F_1 \) is the unique top-ready inner face of \( C_2 \). Let \( v \) be the lower-left vertex of \( F_1 \). Since \( v \) is either w-missing or s-missing, we have the following two cases.

**Case 1:** \( v \) is w-missing

As shown in Fig. 29(a), the lower-right vertex of \( F_0 \) is located below \( v \). If we assume otherwise, then it contradicts the fact that \( C \) is a canonical floorplan. Hence, \( F_0 \) is not ready in \( C_2 \). On the other hand, \( F_0 \) is top-ready. We note that the other inner faces adjacent to the outer face in \( C_2 \) are not ready.

**Case 2:** \( v \) is s-missing

As shown in Fig. 29(b), the upper-left vertex of \( F_0 \) is located to the left of \( v \). Hence, \( F_0 \) is not ready in \( C_2 \). On the other hand, \( F_0 \) is top-ready. Similarly, the other inner faces adjacent to the outer face in \( C_2 \) are not ready.

Therefore, the claim holds for \( C_1 \). If we assume that the claim holds for \( C_i \), then \( C_i \) has the unique top-ready inner face \( F_i \) and no other ready inner faces. By a similar observation for \( C_1 \), the claim holds for \( C_{i+1} \).

**Lemma 7.2** implies that if we know (1) the type of \( F_i \), and (2) if the lower-left vertex of \( F_i \) is either w-missing or s-
missing, then we can reconstruct $C_i$ from $C_{i+1}$. Recall that each $F_i$ is classified into one of four types, as discussed in Section 4.1. Hence, if we store such information for each $i = 2, 3, \ldots, f - 1$, then the original floorplan can be reconstructed. We use two bits for (1) and one bit for (2). Thus we use three bits for each face, except for $F_f$. See Fig. 28 for an example. In this example, types 0, 1, 2, and 3 are represented by "00," "01," "10," and "11," respectively (the four types are shown in Fig. 12). For the lower-left vertex $v$ of $F_i$, 0 indicates that $v$ is s-missing, and 1 represents otherwise. We define the code for a given canonical floorplan as the concatenation of the codes for the inner faces $F_i$, $i = 2, 3, \ldots, f$. The code for the canonical floorplan $C$ in Fig. 28 is "011 110 111 000 101 100 000." Thus the total length of the code is $3f - 3$ bits. If we regard this code as one for a mosaic floorplan, then we have the following theorem.

**Theorem 7.3 (112).** One can encode any mosaic floorplan with $f$ inner faces by $3f - 3$ bits. Coding and decoding each take $\Theta(f)$ time.

8. Representation for Rectangle Packings: Sequence-Pair

In this section, we consider the well-known sequence-pair representation for rectangles on a plane. A rectangle packing is a set of rectangles on a large rectangle, and a class of rectangle packings contains the class of floorplans. Hence, the sequence-pair that represents rectangle packings can be applied to floorplans. Originally, Murata et al. [21] proposed a sequence-pair for representing a packing of modules on a chip.

Let $\Pi$ be a packing of rectangles on a large rectangle $D$. Figure 30 shows an example of a packing of rectangles. In VLSI, each rectangle and the large rectangle correspond to a module and a chip, respectively. Now we consider a sequence-pair for $\Pi$. Let $R$ be a floorplan such that $R$ is a partition (dissection) of $D$, each inner face contains at most one rectangle in $\Pi$, and every rectangle in $\Pi$ is contained in an inner face of $R$. For example, Fig. 31 shows seven rectangles in a floorplan with ten inner faces.

To represent relationships between rectangles, we define the loci for rectangles containing a rectangle. The upper-right locus of a rectangle $F$ containing a rectangle is an alternative sequence $(r_1, r_2, \ldots, r_p)$ of horizontal and vertical line segments. The horizontal line segment $r_1$ starts at the center of $F$ and ends when it intersects the right line segment of $F$. If $i$ is even, then $r_i$ is the vertical line segment that starts at the endpoint of $r_{i-1}$ and ends at an n-missing vertex. Otherwise, $i$ is odd, and $r_i$ is the horizontal line segment that starts at the endpoint of $r_{i-1}$ and ends at an e-missing vertex. The segment $r_p$ ends at the upper-right vertex of the outer face. The lower-left, upper-left, and lower-right loci are defined in a similar way. All the loci of Fig. 31 are illustrated in Figs. 32(a) and (b). Intuitively, the upper-right (lower-left, upper-left, and lower-right) locus of $F$ is a route along the line segments of a floorplan to its upper-right (lower-left, upper-left, and lower-right) vertex.

The positive locus of $F$ is the union of the lower-left and upper-right loci. Similarly, the negative locus of $F$ is the union of the upper-left and lower-right loci. If we regard the lower-left vertex of a floorplan as the origin, the south line segment of the outer face as x-axis, and the west line segment of the outer face as y-axis, then the positive locus of $F$ is a monotonically increasing function and the negative locus of $F$ is a monotonically decreasing function. The positive and negative loci are uniquely defined for each rectangle. We have the following property.
**Lemma 8.1** ([21]). \textit{Positive loci do not intersect. Similarly, negative loci do not intersect.}

By Lemma 8.1, the positive loci (negative loci) can be ordered linearly from the upper-left to the lower-right (from the lower-left to the upper-right). We define \( \Gamma_+ \) to be the sequence obtained by arranging the rectangles following the order of the positive loci, and, similarly, we define \( \Gamma_- \) to be the sequence obtained by arranging the rectangles following the order of the negative loci. A \textit{sequence-pair} \((\Gamma_+, \Gamma_-)\) is a pair of two sequences \(\Gamma_+\) and \(\Gamma_-\). For example, the sequence-pair of the rectangle packing in Fig. 30 is \((\Gamma_+, \Gamma_-) = (abcdfg, dcbaef)\). Each rectangle in a sequence-pair uses \(\log_2 f\) bits. If a sequence-pair has \(f\) rectangles, then a sequence-pair takes at least \(2f/\log_2 f\) bits.

Now we discuss how to decode a sequence-pair in order to reconstruct a packing. Obviously, a sequence-pair represents the combinatorial positional relationship between any two rectangles, and it does not represent the coordinates of any rectangle on a large rectangle. In the following, we explain in detail the information that is stored in a sequence-pair. Let \((\Gamma_+, \Gamma_-)\) be a sequence-pair, and let \(F_1, F_2\) be two rectangles in a packing. If \(F_1\) appears before \(F_2\) in \(\Gamma_+\) and \(F_1\) appears before \(F_2\) in \(\Gamma_-\), then the positive locus of \(F_1\) appears above the positive locus of \(F_2\) and the negative locus of \(F_1\) appears below the negative locus of \(F_2\). Hence, we say that \(F_2\) is located to the east of \(F_1\). If \(F_1\) appears before \(F_2\) in \(\Gamma_+\) and \(F_1\) appears after \(F_2\) in \(\Gamma_-\), then we say that \(F_2\) is located to the south of \(F_1\). If \(F_1\) appears after \(F_2\) in \(\Gamma_+\) and \(F_1\) appears before \(F_2\) in \(\Gamma_-\), then we say that \(F_2\) is located to the north of \(F_1\). If \(F_1\) appears after \(F_2\) in \(\Gamma_+\) and \(F_1\) appears after \(F_2\) in \(\Gamma_-\), then we say that \(F_2\) is located to the west of \(F_1\). A sequence-pair represents the above geometric relationships between any two rectangles.

A \((\Gamma_+, \Gamma_-)\)-packing is a packing that satisfies the geometric relationships represented by a sequence-pair \((\Gamma_+, \Gamma_-)\). Figure 33 is useful for understanding these relationships. Each dotted line from the lower left corresponds to a rectangle in \(\Gamma_+\), and each dotted line from upper left corresponds to a rectangle in \(\Gamma_-\). The rectangles are arranged in the intersection of the two dotted lines corresponding to the rectangles on the large rectangle. We note that a dotted line from the lower left represents a positive locus of a rectangle, and a dotted line from the upper left represents a negative locus of a rectangle. We note that a figure such as Fig. 33 is a \((\Gamma_+, \Gamma_-)\)-packing.

In the original problem in [21], the width and height of each rectangle are given. If the width and height are designated for each rectangle, we can define an “optimal” packing. A \((\Gamma_+, \Gamma_-)\)-packing is \textit{optimal} if it has the minimum width and height among all \((\Gamma_+, \Gamma_-)\)-packings. Murata et al. [21] proposed a method that compute the optimal \((\Gamma_+, \Gamma_-)\)-packing. We briefly explain their method. A horizontal-constraint graph \(G_{H}(V_H, E_H)\) of rectangles is a graph with a vertex set \(V_H\) and an edge set \(E_H\). \(V_H\) is a set of a source \(s\), sink \(t\), and \(f\) vertices, corresponding to the rectangles in a packing. \(E_H\) is a set of \((s, v)\) and \((v, t)\) for each \(v \in V_H\), and \((u, v)\) if and only if \(u\) is located to the west of
v. For each \( v \in V_H \setminus \{s, t\} \), the width is assigned as its vertex-weight. The weights of \( s \) and \( t \) are both zero. Similarly, a vertical-constraint graph \( G_V(V_V, E_V) \) of rectangles is defined. Both graphs are directed acyclic graphs. The \( x \)- and \( y \)-coordinates of each rectangle are determined by assigning the longest path length between the source \( s \) to a vertex corresponding to the rectangle in \( G_H \) and \( G_V \), respectively. Similarly, the width and height of the large rectangle is determined by computing the longest paths between \( s \) and \( t \) in \( G_H \) and \( G_V \). The obtained packing is an optimal \((\Gamma_+, \Gamma_-)\)-packing.

9. Conclusions

We have surveyed recent developments in floorplan representations. This survey contains several codes for floorplans, including codes for floorplans with edge lengths and mosaic floorplans. In addition, we briefly introduced the well-known sequence-pair representation for rectangle packings.

A variety of representations for floorplans have been proposed, and there are many interesting problems that remain to be solved. For a floorplan with \( f \) inner faces, the most compact code uses at most \( 4f \) bits [26, 31]. On the other hand, the upper bound on the number of floorplans with \( f \) inner faces (see Theorem 2.1) implies that there are codes that require less than or equal to \( 3.75f \) bits. Thus, there is a gap between the attained compactness and the theoretical bound. By applying a partitioning approach [2, 10], we may be able to design a representation for which the memory size is asymptotically optimal. This approach uses the exhaustive table for “small” floorplans. Then we have two question. Can we design an optimal representation by applying a partitioning approach? Can we design a code that takes \( \alpha f \) bits for some constant \( \alpha < 4 \) without a table?

An L-floorplan is a partition (dissection) of a rectangle into smaller rectangles or L-shape polygons. L-floorplans are a wider class that contains a class of floorplans, and it can be considered as an extension of floorplans. Recently, compact codes for L-floorplans have been proposed by Karim et al. [18]. They proposed two codes that take \( 6f + 3L + 2n_2 - 2 \) bits and \( 5f + 6L - 4 \) bits, respectively, where \( f \) is the number of inner faces, \( L \) is the number of L-shape polygons, and \( n_2 \) is the number of vertices of degree two. Coding and decoding each take \( \Theta(f) \) time.

A slicing floorplan is a mosaic floorplan with a “slice structure.” A slicing floorplan can be constructed by repeatedly slicing by (1) a vertical line segment touching the north line segment of the outer face and the south line segment of the outer face, and (2) a horizontal line segment touching the west line segment of the outer face and the east line segment of the outer face. See [25, 29] for a definition of slicing floorplans. A class of slicing floorplans are contained in a class of mosaic floorplans. Slicing floorplans have many applications, including VLSI layout design [32]. Coding slicing floorplans is one of the interesting problems that remain to be solved.

Acknowledgment

We greatly thank the anonymous referees and editors for their fruitful comments and suggestions.

REFERENCES
